

SOP for Writing Node.js Code

Here's a collection of templates, tools, and resources to enforce the SOP for writing consistent Node.js code:

- **Install these Dependencies**

1. **Http-errors** ([http-errors](#) module is a utility for creating custom HTTP error objects. It simplifies generating meaningful error responses with proper HTTP status codes and descriptive messages)
2. **Express** (Express.js is a fast, minimalist web application framework for Node.js. It provides robust features to build web applications and APIs easily and efficiently.)
3. **Cookie-parser** ([cookie-parser](#) is a middleware for Node.js applications, commonly used with Express.js, to parse and manage cookies in HTTP requests. It simplifies handling cookies by converting them into an accessible [req.cookies](#) object)
4. **Morgan** (Morgan is a popular HTTP request logger middleware for Node.js, commonly used in Express.js applications. It helps log incoming requests to the server, providing useful information such as the HTTP method, URL, status code, and response time. This logging is especially useful for debugging, monitoring, and analyzing web traffic.)
5. **Connect-flash** ([connect-flash](#) is a middleware for Node.js applications, commonly used with Express.js, that enables the storage and retrieval of flash messages. Flash messages are temporary messages stored in the session, typically used to convey information or errors to users after a redirect. This functionality is especially useful for providing feedback to users following actions like form submissions or login attempts.)
6. **Md5** (MD5 is a widely used cryptographic hash function that produces a 128-bit hash value, typically represented as a 32-character hexadecimal number. It was designed by Ronald Rivest in 1991 to replace an earlier hash function, MD4, and was specified in 1992 as RFC 1321.)
7. **Multer** (Multer is a middleware for Node.js applications, commonly used with Express.js, that simplifies handling [multipart/form-data](#)—the encoding type used for forms that include file uploads. It processes incoming requests containing files and makes them accessible via the [req.file](#) or [req.files](#) objects.)
8. **Compression** (Compression refers to the process of reducing the size or volume of an object or substance by applying pressure. This concept is

applicable across various fields, each with its specific context and applications.
)

9. **Helmet** (Helmet is a middleware designed to enhance security by setting various HTTP headers. These headers help protect against common web vulnerabilities such as Cross-Site Scripting (XSS), Clickjacking, and other attacks.)

-> Package-lock.json:-

```
node_modules/@babel/core": {
  "version": "7.26.0",
  "resolved": "https://registry.npmjs.org/@babel/core/-/core-7.26.0.tgz",
  "integrity":
"sha512-i1SLeK+DzNnQ3LL/CswPCa/E5u4lh1k6IAEphON8F+cXt0t9euTshDru0q7/lqMa1PMPz
5RnHuHscF8/ZJsStg==",
  "dev": true,
  "dependencies": {
    "@ampproject/remapping": "^2.2.0",
    "@babel/code-frame": "^7.26.0",
    "@babel/generator": "^7.26.0",
    "@babel/helper-compilation-targets": "^7.25.9",
    "@babel/helper-module-transforms": "^7.26.0",
    "@babel/helpers": "^7.26.0",
    "@babel/parser": "^7.26.0",
    "@babel/template": "^7.25.9",
    "@babel/traverse": "^7.25.9",
    "@babel/types": "^7.26.0",
    "convert-source-map": "^2.0.0",
    "debug": "^4.1.0",
    "gensync": "^1.0.0-beta.2",
    "json5": "^2.2.3",
    "semver": "^6.3.1"
  },
  "engines": {
    "node": ">=6.9.0"
  },
  "funding": {
    "type": "opencollective",
```

```

    "url": "https://opencollective.com/babel"
  }
},
"node_modules/@babel/core/node_modules/debug": {
  "version": "4.4.0",
  "resolved": "https://registry.npmjs.org/debug/-/debug-4.4.0.tgz",
  "integrity":
"sha512-6WTZ/lyCY/T6BALoZHaE4ctp9xm+Z5kY/pzYaCHRFeyVhojxIrm+46y68HA6hr0TcwEss
oxNiDEUJQjFPZ/RYA==",
  "dev": true,
  "dependencies": {
    "ms": "^2.1.3"
  },
  "engines": {
    "node": ">=6.0"
  },
  "peerDependenciesMeta": {
    "supports-color": {
      "optional": true
    }
  }
},
"node_modules/@babel/core/node_modules/ms": {
  "version": "2.1.3",
  "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
  "integrity":
"sha512-6FlzubTLZG3J2a/NVCAleEhHzq5oxgHyaCU9yYXvcLsvoVaHJq/s5xXI6/XXP6tz7R9xA
OtHnSO/tXtF3WRTIA==",
  "dev": true
},
"node_modules/@babel/core/node_modules/semver": {
  "version": "6.3.1",
  "resolved": "https://registry.npmjs.org/semver/-/semver-6.3.1.tgz",
  "integrity":
"sha512-BR7VvDCVHO+q2xBEWskxS6DJE1qRnb7DxzUrogb71CWoSficBxYsiAGd+KI0mmq/
MprG9yArRkyrQxTO6XjMzA==",
  "dev": true,
  "bin": {
    "semver": "bin/semver.js"
  }
},
"node_modules/@babel/generator": {
  "version": "7.26.5",
  "resolved": "https://registry.npmjs.org/@babel/generator/-/generator-7.26.5.tgz",

```

```

    "integrity":
"sha512-2caSP6fN9I7HOe6nqhtft7V4g7V/gfDsC3Ag4W7kEzzvRGKqiv0pu0HogPiZ3KaVSoND
hUws6lJjDjpfmYIXw==",
    "dev": true,
    "dependencies": {
      "@babel/parser": "^7.26.5",
      "@babel/types": "^7.26.5",
      "@jridgewell/gen-mapping": "^0.3.5",
      "@jridgewell/trace-mapping": "^0.3.25",
      "jsgc": "^3.0.2"
    },
    "engines": {
      "node": ">=6.9.0"
    }
  },
  "node_modules/@babel/helper-annotate-as-pure": {
    "version": "7.25.9",
    "resolved":
"https://registry.npmjs.org/@babel/helper-annotate-as-pure/-/helper-annotate-as-pure-7.25.9.tgz
",
    "integrity":
"sha512-gv7320KBUFJz1Rnyllg5WWYPRXKZ884AGkYpgpWW02TH66DI+HaC1t1CKd0z3R4b
6hdYEcmrNZHUmfCP+1u3/g==",
    "dev": true,
    "dependencies": {
      "@babel/types": "^7.25.9"
    },
    "engines": {
      "node": ">=6.9.0"
    }
  },
  "node_modules/@babel/helper-compilation-targets": {
    "version": "7.26.5",
    "resolved":
"https://registry.npmjs.org/@babel/helper-compilation-targets/-/helper-compilation-targets-7.26.5
.tgz",
    "integrity":
"sha512-IXuyn5EkouFJscIDuFF5EsiSolseme1s0CZB+QxVugqJLYmKdxl1VfIBOst0SUu4rnk2Z7
kqTwmoO1lp3HlfnA==",
    "dev": true,
    "dependencies": {
      "@babel/compat-data": "^7.26.5",
      "@babel/helper-validator-option": "^7.25.9",
      "browserslist": "^4.24.0",

```

```

    "lru-cache": "^5.1.1",
    "semver": "^6.3.1"
  },
  "engines": {
    "node": ">=6.9.0"
  }
},
"node_modules/@babel/helper-compilation-targets/node_modules/lru-cache": {
  "version": "5.1.1",
  "resolved": "https://registry.npmjs.org/lru-cache/-/lru-cache-5.1.1.tgz",
  "integrity":
"sha512-KpNARQA3Iwv+jTA0utUVVbrh+Jlrr1Fv0e56GGzAFOXN7dk/FviaDW8LHmK52Dlch4
WP2n6gl8vN1aesBFgo9w==",
  "dev": true,
  "dependencies": {
    "yallist": "^3.0.2"
  }
},
"node_modules/@babel/helper-compilation-targets/node_modules/semver": {
  "version": "6.3.1",
  "resolved": "https://registry.npmjs.org/semver/-/semver-6.3.1.tgz",
  "integrity":
"sha512-BR7VvDCVHO+q2xBEWskxS6DJE1qRnb7DxzUrogb71CWoSficBxYsiAGd+KI0mmq/
MprG9yArRkyrQxTO6XjMzA==",
  "dev": true,
  "bin": {
    "semver": "bin/semver.js"
  }
},
"node_modules/@babel/helper-compilation-targets/node_modules/yallist": {
  "version": "3.1.1",
  "resolved": "https://registry.npmjs.org/yallist/-/yallist-3.1.1.tgz",
  "integrity":
"sha512-a4UGQaWPH59mOXUYnAG2ewncQS4i4F43Tv3JoAM+s2VDAmS9NsK8GpDMLrCH
PksFT7h3K6TOoUNn2pb7RoXx4g==",
  "dev": true
},
"node_modules/@babel/helper-create-class-features-plugin": {
  "version": "7.25.9",
  "resolved":
"https://registry.npmjs.org/@babel/helper-create-class-features-plugin/-/helper-create-class-feat
ures-plugin-7.25.9.tgz",

```

```

    "integrity":
"sha512-UTZQMvt0d/rSz6KI+qdu7GQze5TlajwTS++GUozlw8VBJDEOAqSXwm1WvmYEWzwd
qSGQshRocPDqrt4HBZB3fQ==",
    "dev": true,
    "dependencies": {
      "@babel/helper-annotate-as-pure": "^7.25.9",
      "@babel/helper-member-expression-to-functions": "^7.25.9",
      "@babel/helper-optimise-call-expression": "^7.25.9",
      "@babel/helper-replace-supers": "^7.25.9",
      "@babel/helper-skip-transparent-expression-wrappers": "^7.25.9",
      "@babel/traverse": "^7.25.9",
      "semver": "^6.3.1"
    },
    "engines": {
      "node": ">=6.9.0"
    },
    "peerDependencies": {
      "@babel/core": "^7.0.0"
    }
  },
  "node_modules/@babel/helper-create-class-features-plugin/node_modules/semver": {
    "version": "6.3.1",
    "resolved": "https://registry.npmjs.org/semver/-/semver-6.3.1.tgz",
    "integrity":
"sha512-BR7VvDCVHO+q2xBEWskxS6DJE1qRnb7DxzUrogb71CWoSficBxYsiAGd+KI0mmq/
MprG9yArRkyrQxTO6XjMzA==",
    "dev": true,
    "bin": {
      "semver": "bin/semver.js"
    }
  },
  "node_modules/@babel/helper-create-regexp-features-plugin": {
    "version": "7.26.3",
    "resolved":
"https://registry.npmjs.org/@babel/helper-create-regexp-features-plugin/-/helper-create-regexp-f
eatures-plugin-7.26.3.tgz",
    "integrity":
"sha512-G7ZRb40uUgdKOQqPLjFD12ZmGA54PzqDFUv2BKImnC9QIfGhIHKvVML0oN8IUIDq4
iRqpq74ABpvOaerfWdong==",
    "dev": true,
    "dependencies": {
      "@babel/helper-annotate-as-pure": "^7.25.9",
      "regexpu-core": "^6.2.0",
      "semver": "^6.3.1"
    }
  }
}

```

```

    },
    "engines": {
      "node": ">=6.9.0"
    },
    "peerDependencies": {
      "@babel/core": "^7.0.0"
    }
  },
  "node_modules/@babel/helper-create-regexp-features-plugin/node_modules/semver": {
    "version": "6.3.1",
    "resolved": "https://registry.npmjs.org/semver/-/semver-6.3.1.tgz",
    "integrity":
    "sha512-BR7VvDCVHO+q2xBEWskxS6DJE1qRnb7DxzUrogb71CWoSficBxYsiAGd+Kl0mmq/
    MprG9yArRkyrQxTO6XjMzA==",
    "dev": true,
    "bin": {
      "semver": "bin/semver.js"
    }
  },
  "node_modules/@babel/helper-define-polyfill-provider": {
    "version": "0.6.3",
    "resolved":
    "https://registry.npmjs.org/@babel/helper-define-polyfill-provider/-/helper-define-polyfill-provider-
    0.6.3.tgz",
    "integrity":
    "sha512-HK7Bi+Hj6H+VTHA3ZvBis7V/6hu9QuTrnMXNybfUf2iiuU/N97I8VjB+KbhFF8Rld/Lx5M
    zoCwPCpPjfk+n8Cg==",
    "dev": true,
    "dependencies": {
      "@babel/helper-compilation-targets": "^7.22.6",
      "@babel/helper-plugin-utils": "^7.22.5",
      "debug": "^4.1.1",
      "lodash.debounce": "^4.0.8",
      "resolve": "^1.14.2"
    },
    "peerDependencies": {
      "@babel/core": "^7.4.0 || ^8.0.0-0 <8.0.0"
    }
  }
}

```

- **Folder Structure Template**

Create the following directory structure:

```
|— II
|   |— apis
|   |— bin
|   |— controllers
|   |— middleware
|   |— modals
|   |— json
|   |— uploads
|   |— node_modules
|   |— public
|   |— routes
|   |— views
|   |— app.js
|   |— auth.js
|   |— package.json
|   |— auth.js
|   |— package-lock.json
|— .env
|— README.md
|— .gitignore
```

- Example app.js:

```
var createError = require("http-errors");
var express = require("express");
var path = require("path");
var cookieParser = require("cookie-parser");
var logger = require("morgan");
var flash = require("connect-flash");
const multer = require("multer")
const md5 = require("md5");
const session = require("express-session");
const jwt = require("jsonwebtoken");
const compression = require("compression");
const MainBilling = require("./routes/MainBilling");
const Reports = require("./routes/repots");
const helmet = require("helmet");
const concentRouter = require("./routes/concent");
```



```

const appRouter = require("./routes/AppConfig");
const notes = require("./routes/notes");

const { Admin, KYC } = require("./models/Kyc");
const clinicRoutes = require("./routes/clinicalRoute");
var adminInventory = require("./routes/adminInventory");
var usersRouter = require("./routes/users");
const { radioRouter } = require("./routes/Radiology");
var patientconfigRouter = require("./routes/patientconfig");
const mainInv = require("./routes/mainInv");
var billing = require("./routes/billing");
var indexRouter = require("./routes/Admin");
// var patRegRoute = require("./routes/PatientRegistration");
var packageConfigRouter = require("./routes/packageConfig");
var pathologyRouter = require("./routes/pathology");
var dashboard = require("./routes/360");
// const MainBilling = require("./routes/MainBilling");
const findpatient = require("./routes/PatientRegistration");
var embRouter = require("./routes/embrology");
var fieldData = require("./routes/fieldData");
const { ClinicConfiguration } = require("./models/clinicConfig");
const { UserTokens } = require("./models/UserTokens");
const CryptoJS = require("crypto-js");
function decryptData(encryptedData, secretKey) {
  try {
    const bytes = CryptoJS.AES.decrypt(encryptedData, secretKey);
    const decryptedText = bytes.toString(CryptoJS.enc.Utf8);

    if (!decryptedText) {
      throw new Error("Decryption failed, possibly due to incorrect key or missing salt/IV.");
    }
    return decryptedText;
  } catch (error) {
    console.error("Decryption error:", error.message);
    return null;
  }
}

const {
  kycCtrl,
  dashboardCtrl,

```

```

    modifyApt,
    modifyVisit,
    apt1,
    apt2,
    apt3,
    apt4,
    login,
    logoutFromEverywhere,
    register,
    logout,
    getAllKYCDetails,
    fetchFU,
    fetchApt,
    fetchVisits,
    fetchConv,
  } = require("./controllers/controller");

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "public/images/myuploads");
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    cb(
      null,
      "clinical-" +
        file.fieldname +
        "-" +
        uniqueSuffix +
        path.extname(file.originalname)
    );
  },
});

const upload = multer({ storage: storage });

var app = express();

app.use(cookieParser());

app.use(
  session({
    secret: "abc", // change 'your_secret_key' to a real secret
    resave: false,
  })
);

```

```
    saveUninitialized: true,  
    cookie: { secure: "auto" }, // Adjust settings as necessary  
  })  
);
```

```
app.use(logger("dev"));  
app.use(express.json());  
app.use(express.urlencoded({ extended: true }));  
app.use(express.static(path.join(__dirname, "public")));
```

```
app.use(  
  compression({  
    level: 6,  
    threshold: 100 * 1000,  
    filter: (req, res) => {  
      if (req.header["x-no-compression"]) {  
        return false;  
      }  
      return compression.filter(req, res);  
    },  
  })  
);
```

```
const authenticateMiddleware = (req, res, next) => {  
  const token = req.cookies.token;
```

```
  // Allow access to the /1 and / routes  
  const publicRoutes = ['/1', '/', '/login', '/getUserClinic', '/logoutFromEverywhere',];
```

```
  if (publicRoutes.includes(req.path)) {  
    return next(); // Skip the authentication check for these routes  
  }
```

```
  if (!token) {  
    return res.status(404).send('Not Found'); // Return 404 error if token is missing  
  }
```

```
  next();  
};
```

```
// Place this at the top of your app.js file  
app.use(authenticateMiddleware);
```

```

// app.use(helmet());

// Middleware to ensure authentication
// const ensureAuthenticated = (req, res, next) => {
//   // Routes that do not require authentication
//   const openRoutes = ['/1', '/login'];

//   // Skip authentication check for open routes
//   if (openRoutes.includes(req.path)) {
//     return next();
//   }

//   console.log(req.session)
//   // Redirect unauthenticated users to login
//   if (!req.session.user) {

//     return res.redirect('/1');
//   }

//   // Proceed if authenticated
//   next();
// };

// // Apply ensureAuthenticated globally
// app.use(ensureAuthenticated);

const JWT_SECRET = "ll";
const verifyToken = async (req, res, next) => {
  const token = req.cookies.token; // Read token from HttpOnly cookie
  console.log(token);

  if (!token) {
    return res.redirect("/1");
  }
  const user = await UserTokens.findOne({ where: { jwtToken: token } });
  console.log(user);

  if (!user) {
    return res.redirect("/1");
  }
  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded; // Attach user details to req.user
    res.locals.user = req.user; // Make user available in templates
  }

```

```

    next();
  } catch (err) {
    return res.status(401).json({ message: "Invalid Token" });
  }
};

function checkRights(requiredRight) {
  return function (req, res, next) {
    // Check if the user exists and has the rights array
    if (req.user && req.user.rights && Array.isArray(req.user.rights)) {
      // Check if the requiredRight is included in the rights array
      if (req.user.rights.includes(requiredRight)) {
        // User has the required right
        return next();
      }
    }
  }
}

// If the required right is not found, redirect or handle access denial
return res.redirect("/1"); // Redirecting to the access denied page
};
}

```

```

app.use((req, res, next) => {
  const token = req.cookies.token;

  console.log(token);
  if (token) {
    try {
      const decoded = jwt.verify(token, JWT_SECRET);
      req.user = decoded;
      res.locals.user = req.user;
    } catch (err) {
      console.error("Invalid token", err);
    }
  }
  next();
});

```

```

// Middleware to get today's date
const todayDateMiddleware = (req, res, next) => {
  const today = new Date();
  const formattedDate = today.toISOString().split("T")[0]; // Format as YYYY-MM-DD
  req.todayDate = formattedDate;
  next();
}

```

```

};
app.use(todayDateMiddleware);

// Set views and engine
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");

app.use((req, res, next) => {
  if (req.user) {
    console.log("User given: ", req.user);
    res.locals.clinicName = req.user.clinicName || "X";
  } else {
    console.log("No user found");
    res.locals.clinicName = "X"; // Default clinic name when no user is found
  }
  next();
});

// Routes setup
app.use("/field", fieldData);
app.use("/adminInv", verifyToken, checkRights('51'), adminInventory);
app.use("/clinic", verifyToken, checkRights('2'), clinicRoutes);
app.use("/users", verifyToken, usersRouter);
app.use("/radiology", verifyToken, checkRights('3'), radioRouter);
app.use("/patient", verifyToken, checkRights('3'), patientconfigRouter);
app.use("/MainInv", verifyToken, checkRights('52'), mainInv);
app.use("/billing", verifyToken, checkRights('23'), billing);
app.use("/Admin", verifyToken, checkRights('0000'), indexRouter);
app.use("/package", verifyToken, checkRights('2'), packageConfigRouter);
app.use("/concent", checkRights('3'), concentRouter);
app.use("/pathology", verifyToken, checkRights('3'), pathologyRouter);
app.use("/main", verifyToken, dashboard);
app.use("/findpatient", verifyToken, checkRights('3'), findpatient);
app.use("/MainBill", verifyToken, checkRights('23'), MainBilling);
app.use("/embrology", verifyToken, checkRights('174'), embRouter);
app.use("/appConfig", verifyToken, appRouter);
// app.use("/MainBill", verifyToken, checkRights('23'), MainBilling);
app.use("/reports", verifyToken, Reports);

app.get("/getUserClinic", async (req, res) => {

  try {
    // Log the user object to see the details (this is optional, for debugging)
    console.log('Req', req.query);
  }
});

```

```
// Extract the user identifier (e.g., userEmail, userId) from req.user
const { username } = req.query; // You could also use req.user.id if userEmail isn't
provided
```

```
// Step 1: Query the User schema (or table) to get the clinicId
const user = await SuperAdmin.findOne({
  where: { username: username }, // or you can use req.user.id to look up by userId
});
```

```
// If user is not found, return an error message
if (!user) {
  return res.status(404).json({ msg: "User not found." });
}
```

```
// Extract the clinicId from the user record
const { clinicId } = user;
```

```
// Step 2: Use the clinicId to find the clinic from ClinicConfiguration table
const clinic = await ClinicConfiguration.findAll({
  where: { clinic_id: clinicId },
});
console.log(clinic)
// If clinic is not found, return an error message
if (!clinic) {
  return res.status(404).json({ msg: "Clinic not found." });
}
```

```
// Return the clinic details
res.status(200).json(clinic);
} catch (error) {
  console.error("Error fetching clinic details:", error);
  res.status(500).json({
    msg: "An error occurred while fetching clinic details.",
  });
}
});
```

```
// Routes
app.get("/home", verifyToken, async (req, res) => {
  console.log(req.query);
  const { kyc_id } = req.query;
  console.log(kyc_id);
});
```

```

const declId = decryptData(kyc_id, "ll");
const admins = await Admin.findAll();
try {
  if (declId) {
    const kycData = await KYC.findByPk(declId);

    if (!kycData) {
      console.error("record not found:", declId);
      return res.status(404).render("error", { message: "record not found" });
    }

    const kycValues = kycData.get({ plain: true });
    console.log(" values:", kycValues);
    return res.render("form-new", {
      a: kycValues,
      loginName: req.user.username,
      admins,
    });
  } else {
    console.log(req.user); // Now using JWT to fetch user info
    res.render("form-new", { a: "", loginName: req.user.username, admins });
  }
} catch (error) {
  console.error("Error fetching data:", error);
  return res
    .status(500)
    .render("error", { message: "Internal Server Error" });
}
});

app.get("/", (req, res) => {
  res.redirect("/1");
});

app.get("/1", async (req, res) => {
  const token = req.cookies.token; // Retrieve the JWT token from cookies

  if (token) {
    try {
      // Verify the token
      const decoded = jwt.verify(token, JWT_SECRET);
      const user = await UserTokens.findOne({ where: { jwtToken: token } });
      console.log("hi");
      console.log(user);
    }
  }
});

```



```

    if (user) {
      return res.redirect("/patientlist");
    } else {
      console.log("in");
      return res.render("login");
    }
    // If the token is valid, redirect to home
  } catch (err) {
    console.error("Invalid token", err);
    // If token verification fails, continue to render login
  }
}

// If no valid token is found, render the login page
res.render("login");
});

//
const setClinicName = (req, res, next) => {
  const token = req.cookies.token;

  if (token) {
    jwt.verify(token, JWT_SECRET, (err, decoded) => {
      if (!err) {
        res.locals.clinicName = decoded.clinicName; // Set clinicName in res.locals
      }
    });
  }
  next();
};

app.get("/patientlist", verifyToken, setClinicName, async (req, res) => {
  // Get the clinicName from the query parameters
  const clinicName = req.query.clinicName;
  // console.log("session id: ", req.session);

  // Check if the clinicName exists in the query string
  if (clinicName && clinicName.trim() !== "") {
    console.log("Clinic Name Received:", clinicName);
    // Set the clinicName in the session
    req.session.clinicName = clinicName;
  }
}

```

```

// Render the form-new-list template
res.render("form-new-list", {
  user: req.user,
  clinic: req.session.clinicName || "X",
});
});

app.get("/2", (req, res) => {
  res.redirect(301, "/1");
});

app.get("/auditDash", (req, res) => {
  res.render("1-audit-trial-dashboard");
});
app.get("/corfin", (req, res) => {
  res.render("2-corporate-financial-dashboard");
});
app.get("/dailykpidash", (req, res) => {
  res.render("3-daily-kpi-dashboard");
});
app.get("/hosfindash", (req, res) => {
  res.render("4-hospital-financial-dashboard");
});

app.get("/dashboard", dashboardCtrl);
app.get("/get-pat-dtl", getAllKYCDetails);
const checkUHIDRoutes = require("./routes/360new");
app.use(checkUHIDRoutes);
app.post("/register", register);
app.post("/login", login);
app.post("/logout", logout);
app.post("/kycreg", upload.any(), kycCtrl);
app.post("/modify-apt", modifyApt);
app.post("/modify-vis", modifyVisit);
app.post("/appointment_1", apt1);
app.get("/followUp-data/:kyc_id", fetchFU);
app.get("/appointment-data/:kyc_id", fetchApt);
app.get("/visited-appointment-data/:kyc_id", fetchVisits);
app.get("/converted-appointment-data/:kyc_id", fetchConv);
app.post("/appointment_2", apt2);
app.post("/appointment_3", apt3);
app.post("/appointment_4", apt4);
app.post("/logoutFromEverywhere", logoutFromEverywhere);
app.use("/api", notes);

```

```

// Catch 404 and forward to error handler
app.use(function (req, res, next) {
  next(createError(404));
});

// Error handler
app.use(function (err, req, res, next) {
  res.locals.message = err.message;
  res.locals.error = req.app.get("env") === "development" ? err : {};
  res.status(err.status || 500);
  res.render("error");
});

// Start the server
const { con } = require("./sequelize");
const { SuperAdmin } = require("./models/Admin");

const PORT = 5000;
app.listen(PORT, async () => {
  console.log(`Server started at PORT ${PORT}`);
  await con();
});

module.exports = app;

```

- **Testing with compression:**

```

app.use(
  compression({
    level: 6,
    threshold: 100 * 1000,
    filter: (req, res) => {
      if (req.header["x-no-compression"]) {
        return false;
      }
      return compression.filter(req, res);
    },
  })
);

```

- **Testing with session:**

```
app.use(
  session({
    secret: "abc", // change 'your_secret_key' to a real secret
    resave: false,
    saveUninitialized: true,
    cookie: { secure: "auto" }, // Adjust settings as necessary
  })
);
```

- **Testing with cookie:**

```
app.use((req, res, next) => {
  const token = req.cookies.token;

  console.log(token);
  if (token) {
    try {
      const decoded = jwt.verify(token, JWT_SECRET);
      req.user = decoded;
      res.locals.user = req.user;
    } catch (err) {
      console.error("Invalid token", err);
    }
  }
  next();
});
```

- **Create package.json:**

```
{
  "name": "lifelinkr",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "nodemon ./bin/www",
```

```
"build": "webpack"
},
"dependencies": {
  "@prisma/client": "^5.15.1",
  "axios": "^1.7.9",
  "bcrypt": "^5.1.1",
  "body-parser": "^1.20.2",
  "chai": "^5.1.1",
  "chart.js": "^4.4.7",
  "compression": "^1.7.4",
  "connect-flash": "^0.1.1",
  "cookie-parser": "~1.4.4",
  "cors": "^2.8.5",
  "crypto-js": "^4.2.0",
  "debug": "~2.6.9",
  "dotenv": "^16.4.5",
  "ejs": "^3.1.10",
  "email-verifier": "^0.4.1",
  "exceljs": "^4.4.0",
  "express": "~4.16.1",
  "express-flash": "^0.0.2",
  "express-generator": "^4.16.1",
  "express-session": "^1.18.0",
  "express-validator": "^7.1.0",
  "handlebars": "^4.7.8",
  "helmet": "^7.1.0",
  "http-errors": "~1.6.3",
  "jade": "~1.11.0",
  "jquery": "^3.7.1",
  "jsonwebtoken": "^9.0.2",
  "lifelinkr": "file:",
  "md5": "^2.3.0",
  "mocha": "^10.4.0",
  "mongoose": "^8.4.1",
  "morgan": "~1.9.1",
  "multer": "^1.4.5-lts.1",
  "mysql": "^2.18.1",
  "mysql2": "^3.10.1",
  "nodemon": "^3.1.3",
  "passport": "^0.7.0",
  "passport-local": "^1.0.0",
  "pdf-lib": "^1.17.1",
  "pdfkit": "^0.15.0",
  "pug": "^3.0.3",
```

```

    "qrcode": "^1.5.4",
    "sequelize": "^6.37.3",
    "supertest": "^7.0.0",
    "twilio": "^5.3.4",
    "validator": "^13.12.0"
  },
  "devDependencies": {
    "@babel/core": "^7.26.0",
    "@babel/preset-env": "^7.26.0",
    "babel-loader": "^9.2.1",
    "css-loader": "^7.1.2",
    "mini-css-extract-plugin": "^2.9.2",
    "prisma": "^5.15.1",
    "style-loader": "^4.0.0",
    "webpack": "^5.97.1",
    "webpack-cli": "^6.0.1"
  }
}

```

- **README Template**

Project Name

Description

Briefly describe the project.

Prerequisites

- Node.js >= 14.x
- npm >= 6.x

Installation

1. Clone the repository:

```

``bash
git clone https://github.com/your-repo.git

```

2. Install dependencies:

```

npm install

```

3. Add environment variables in .env file:

PORT=5000

DB_URL=mysql2://localhost:3306/yourdb

- **DATABASE STRUCTURE**

Embriology :--

```
const { DataTypes, INTEGER } = require("sequelize");
const { sequelize } = require("../sequelize");
```

```
const DonorVisit = sequelize.define(
  "DonorVisit",
  {
    donorId: {
      type: DataTypes.STRING,
      primaryKey: true,
      allowNull: false,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    donorVisitNo: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    donorVisitDate: {
      type: DataTypes.DATEONLY,
      allowNull: true,
    },
    clinic: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    uhid: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    donorAccountName: {
      type: DataTypes.STRING,
```

```
    allowNull: false,
  },
  doctorName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  age: {
    type: DataTypes.INTEGER,
    allowNull: true,
  },
  ageRemark: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  totalChildren: {
    type: DataTypes.INTEGER,
    allowNull: true,
    defaultValue: 0, // Set default to 0
  },
  totalMaleChildren: {
    type: DataTypes.INTEGER,
    allowNull: true,
    defaultValue: 0, // Set default to 0
  },
  totalFemaleChildren: {
    type: DataTypes.INTEGER,
    allowNull: true,
    defaultValue: 0, // Set default to 0
  },
  gameteName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  lastRegistrationDate: {
    type: DataTypes.DATEONLY,
    allowNull: true,
  },
  lastPickupDate: {
    type: DataTypes.DATEONLY,
    allowNull: true,
  },
  remarks: {
    type: DataTypes.STRING,
    allowNull: true,
```



```

    },
  },
  {
    tableName: "emb_donor_visits",
    timestamps: true, // Adds createdAt and updatedAt fields automatically
  }
);

```

```

const Donor = sequelize.define(
  "Donor",
  {
    donorId: {
      type: DataTypes.STRING(50),
      allowNull: false,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    code: {
      type: DataTypes.STRING(50),
      allowNull: false,
    },
    eye_color: {
      type: DataTypes.STRING(50),
      allowNull: true,
    },
    skin_color: {
      type: DataTypes.STRING(50),
      allowNull: true,
    },
    hair_color: {
      type: DataTypes.STRING(50),
      allowNull: true,
    },
    bone_structure: {
      type: DataTypes.STRING(50),
      allowNull: true,
    },
    height: {
      type: DataTypes.STRING(50),
      allowNull: true,
    },
    blood_group: {
      type: DataTypes.STRING(50),

```

```

    allowNull: true,
  },
  referral_type: {
    type: DataTypes.STRING(50),
    allowNull: true,
  },
  agency_name: {
    type: DataTypes.STRING(100),
    allowNull: true,
  },
  current_page: {
    type: DataTypes.STRING(100),
    allowNull: true,
  },
  viral_marker: {
    type: DataTypes.STRING(100),
  },
  betaThalassemia: {
    type: DataTypes.STRING,
  },
  chlamydia: {
    type: DataTypes.STRING,
  },
  DFI: {
    type: DataTypes.STRING,
  },
},
{
  timestamps: true,
  underscored: true,
  tableName: "emb_donors",
}
);

```

```

const DonorOocyte = sequelize.define(
  "DonorOocyte",
  {
    donorId: {
      type: DataTypes.STRING(50),
      allowNull: false,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
  },

```

```
dob: {
  type: DataTypes.DATE,
  allowNull: true,
},
code: {
  type: DataTypes.STRING(50),
  allowNull: false,
},
donor_name: {
  type: DataTypes.STRING(100), // New field for donor's name
  allowNull: false,
},
identity_card_type: {
  type: DataTypes.STRING(50), // New field for identity card type
  allowNull: false,
},
identity_card_number: {
  type: DataTypes.STRING(50), // New field for identity card number
  allowNull: false,
},
eye_color: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
skin_color: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
hair_color: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
bone_structure: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
height_feet: {
  type: DataTypes.INTEGER,
  allowNull: true,
},
height_inches: {
  type: DataTypes.INTEGER,
  allowNull: true,
},
```

```
weight: {
  type: DataTypes.FLOAT,
  allowNull: true,
},
bmi: {
  type: DataTypes.FLOAT,
  allowNull: true,
},
blood_group: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
referral_type: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
agency_name: {
  type: DataTypes.STRING(100),
  allowNull: true,
},
viral_marker: {
  type: DataTypes.STRING(100),
},
betaThalassemia: {
  type: DataTypes.STRING,
},
chlamydia: {
  type: DataTypes.STRING,
},
family_health_history: {
  type: DataTypes.TEXT,
  allowNull: true,
},
psychological_evaluation: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
consent_signed: {
  type: DataTypes.BOOLEAN,
  allowNull: false,
  defaultValue: false,
},
current_page: {
  type: DataTypes.STRING(100),
```

```

    allowNull: true,
  },
  status: {
    type: DataTypes.BOOLEAN,
    defaultValue: true,
  },
},
{
  timestamps: true,
  underscored: true,
  tableName: "emb_donors_oocyte",
}
);

```

```

const SemenSample = sequelize.define(
  "SemenSample",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    userId: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
    clinchId: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
    patient_id: {
      type: DataTypes.INTEGER,
    },
    parent: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    Storage: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    status: {
      type: DataTypes.STRING,
      defaultValue: "collected",
    },
  },

```

```

stage: {
  type: DataTypes.STRING,
  defaultValue: "Fresh",
},
date_of_collect: {
  type: DataTypes.DATE,
},
time_of_collect: {
  type: DataTypes.TIME,
},
sexual_abstinence_days: {
  type: DataTypes.INTEGER,
},
ph_value: {
  type: DataTypes.STRING,
},
collected_method: {
  type: DataTypes.STRING,
},
received_on: {
  type: DataTypes.DATE,
  allowNull: false,
},
time_of_receiving: {
  type: DataTypes.TIME,
},
no_of_labels: {
  type: DataTypes.INTEGER,
},
received_by: {
  type: DataTypes.STRING,
},
donorId: {
  type: DataTypes.INTEGER,
},
},
{
  tableName: "embro_semen_sample_collection",
  timestamps: true,
}
);

// SemenSample.sync({alter: true})

```

```
const WashSemenSample = sequelize.define(
  "WashSemenSample",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
    userId: {
      type: DataTypes.INTEGER,
    },
    clinchId: {
      type: DataTypes.INTEGER,
    },
    sample_id: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
    collected_on: {
      type: DataTypes.DATE,
      defaultValue: null,
    },
    patient_id: {
      type: DataTypes.INTEGER,
    },
    mrn: {
      type: DataTypes.STRING,
    },
    fresh: {
      type: DataTypes.STRING,
      defaultValue: null,
    },
    capacitated: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
    capacitatedData: {
      type: DataTypes.JSON,
    },
    volume: {
      type: DataTypes.FLOAT,
      defaultValue: 0.0,
    },
    concentration: {
      type: DataTypes.FLOAT,
```

```
    defaultValue: 0.0,
  },
  total_sperm: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  rapid_progressive: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  slow_progressive: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  non_progressive: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  immobile: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  total_motile_prog: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  normal_in: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  abnormal_in: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  head_defects: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  mid_piece_defects: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  tail_defects: {
    type: DataTypes.FLOAT,
```



```
    defaultValue: 0.0,
  },
  excess_residual_cytoplasm: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  teratoz_index: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  normal_prog_motile: {
    type: DataTypes.FLOAT,
    defaultValue: 0.0,
  },
  semen_appearance: {
    type: DataTypes.STRING,
  },
  homogenize: {
    type: DataTypes.STRING,
  },
  complete_sample: {
    type: DataTypes.STRING,
  },
  diagnosis: {
    type: DataTypes.STRING,
  },
  degree_of_diagnosis: {
    type: DataTypes.STRING,
  },
  vitality_test_method: {
    type: DataTypes.STRING,
  },
  vitality_test_in: {
    type: DataTypes.STRING,
  },
  viscosity: {
    type: DataTypes.STRING,
  },
  washing_technique: {
    type: DataTypes.STRING,
  },
  washing_done_by: {
    type: DataTypes.STRING,
  },
}
```

```
large_halo: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
medium_halo: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
small_halo: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
no_halo: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
fragmented: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
non_fragmented: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
total: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
dfi_percent: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
manual_calculation: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
macs_done: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
hba: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
```

```
dfi: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
dfi_total: {
  type: DataTypes.INTEGER,
},
dfi_total_2: {
  type: DataTypes.INTEGER,
},
picsi: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
general_remarks: {
  type: DataTypes.STRING,
},
private_remarks: {
  type: DataTypes.STRING,
},
bound_motile_sperm: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
non_bound_motile_sperm: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
total_hba: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
hba_positive_sperm_percent: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
hba_negative_sperm_percent: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
total_percent: {
  type: DataTypes.FLOAT,
  defaultValue: 0.0,
},
```

```

    donorId: {
      type: DataTypes.INTEGER,
    },
  },
  {
    tableName: "embro_wash_semen_sample",
    timestamps: true,
  }
);

```

```

// WashSemenSample.sync({alter: true})

```

```

const DiscardedSample = sequelize.define(
  "DiscardedSample",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    userId: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
    clinchId: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
    patient_id: {
      type: DataTypes.INTEGER,
    },
    sample_id: {
      type: DataTypes.INTEGER,
    },
    discardDate: {
      type: DataTypes.DATE,
    },
    discardTime: {
      type: DataTypes.TIME,
    },
    discarded_by: {
      type: DataTypes.STRING,
    },
    reason_for_discard: {
      type: DataTypes.STRING,
    },
  },

```

```

    },
    donorId: {
      type: DataTypes.INTEGER,
    },
  },
  {
    tableName: "embrodiscarded_samples",
    timestamps: true,
  }
);

```

```

const EmbryoTransfer = sequelize.define(
  "EmbryoTransfer",
  {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
    date: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    time: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    embryosTransferred: {
      type: DataTypes.JSON,
      allowNull: true,
      defaultValue: null,
    },
    catheter: {
      type: DataTypes.ENUM(
        "COOK Soft",
        "Embryo Transfer Catheter",
        "Gynetics-Tulip",
        "Hard",
        "IUI Catheter",
        "Labotech",
        "Obturator",
        "Soft",
        "Wallace"
      ),
    },
  },
);

```

```
    allowNull: true,
    defaultValue: null,
  },
  difficulty: {
    type: DataTypes.ENUM("Difficult", "Easy", "Impossible", "Moderate"),
    allowNull: true,
    defaultValue: null,
  },
  transferDFE: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  ecoguided: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  mucous: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  blood: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  repetition: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  bladder: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  allies: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  dilator: {
```

```

    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  comments: {
    type: DataTypes.TEXT,
    allowNull: true,
    defaultValue: null,
  },
  doctor: {
    type: DataTypes.STRING,
    allowNull: true,
    defaultValue: null,
  },
  embryologist: {
    type: DataTypes.STRING,
    allowNull: true,
    defaultValue: null,
  },
  status: {
    type: DataTypes.BOOLEAN,
    allowNull: true,
    defaultValue: false,
  },
  cycleId: {
    type: DataTypes.INTEGER,
    defaultValue: 0,
  },
  patientId: {
    type: DataTypes.INTEGER,
  },
  clinicId: {
    type: DataTypes.INTEGER,
  },
},
{
  timestamps: true, // Automatically adds createdAt and updatedAt fields
  createdAt: "createdAt",
  updatedAt: "updatedAt",
  // sequelize, // We need to pass the connection instance
  modelName: "EmbryoTransfer", // Model name
  tableName: "embryo_transfers", // Table name
}
);

```

```

// const EmbryoTransfer = sequelize.define('EmbryoTransfer', {
//   userId: {
//     type: DataTypes.INTEGER,
//     defaultValue: 1
//   },
//   clinchId: {
//     type: DataTypes.INTEGER,
//     defaultValue: 1
//   },
//   cycleId: {
//     type: DataTypes.INTEGER,
//     defaultValue: 0,
//   },
//   patient_Id: {
//     type: DataTypes.INTEGER,
//   },
//   date: {
//     type: DataTypes.DATEONLY
//   },
//   time: {
//     type: DataTypes.TIME
//   },
//   embryos_id: {
//     type: DataTypes.STRING,
//   },
//   catheter: {
//     type: DataTypes.STRING
//   },
//   difficulty: {
//     type: DataTypes.STRING
//   },
//   transfer_dfe: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   ecoguided: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   repetition: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },

```



```

//   blood: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   mucous: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   bladder: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   allies: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   dialator: {
//     type: DataTypes.BOOLEAN,
//     defaultValue: false
//   },
//   comments: {
//     type: DataTypes.TEXT
//   },
//   doctor: {
//     type: DataTypes.STRING
//   },
//   embryologist: {
//     type: DataTypes.STRING
//   }
// }, {
//   timestamps: true,
//   tableName: 'embryoEmbryoTransfer'
// });

```

```

const CanisterData = sequelize.define(
  "embryo_sperm_freezeData",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    userId: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
  },
  {
    timestamps: true,
    tableName: 'embryoEmbryoTransfer'
  }
);

```

```
},
clinchId: {
  type: DataTypes.INTEGER,
  defaultValue: 1,
},
patient_id: {
  type: DataTypes.INTEGER,
},
sample_id: {
  type: DataTypes.INTEGER,
},
canister: {
  type: DataTypes.STRING,
  allowNull: false,
},
due_date: {
  type: DataTypes.DATEONLY,
  allowNull: false,
},
freeze_in: {
  type: DataTypes.ENUM("vials", "straws"),
  allowNull: false,
},
actual_date_renewal: {
  type: DataTypes.DATEONLY,
  allowNull: false,
},
color: {
  type: DataTypes.STRING,
  allowNull: false,
},
date: {
  type: DataTypes.DATEONLY,
  allowNull: false,
},
done_by: {
  type: DataTypes.STRING,
  allowNull: false,
},
tank: {
  type: DataTypes.STRING,
  allowNull: false,
},
term_duration: {
```

```

    type: DataTypes.DATEONLY,
    allowNull: false,
  },
  time: {
    type: DataTypes.TIME,
    allowNull: false,
  },
  vial_holder: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  vial_position: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  donorId: {
    type: DataTypes.INTEGER,
  },
},
{
  tableName: "embryo_sperm_freezeData", // You can specify the table name if it is different
  timestamps: false, // Disable timestamps if you don't need createdAt/updatedAt fields
}
);

```

// stimulation and culture model

```

const treatmentAdvice = sequelize.define(
  "treatmentAdvice",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
  },

```

```

    treatment: {
      type: DataTypes.JSON,
    },
  },
  {
    tableName: "emb_treatmentadvice",
    alter: true,
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);

```

```

const chiefComplaint = sequelize.define(
  "chiefComplaint",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
    complaint: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    {
      tableName: "emb_chiefcomplaint",
      alter: true,
      timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
    }
  }
);

```

```

const preExistingCondition = sequelize.define(
  "preExistingCondition",

```

```

{
  clinic_id: {
    type: DataTypes.INTEGER,
    defaultValue: 0,
  },
  cycleId: {
    type: DataTypes.INTEGER,
    defaultValue: 0,
  },
  patientId: {
    type: DataTypes.STRING,
  },
  doctorId: {
    type: DataTypes.STRING,
  },
  preCondition: {
    type: DataTypes.JSON,
  },
  Date: {
    type: DataTypes.DATEONLY,
  },
},
{
  tableName: "emb_preexistingcondition",
  alter: true,
  timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
}
);
const allergies = sequelize.define(
  "allergies",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
  },

```

```

    },
    allergies: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    tableName: "emb_allergies",
    alter: true,
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);
// const patientHistory = sequelize.define('patientHistory', {
//   clinic_id:{
//     type:DataTypes.INTEGER,
//     defaultValue:0
//   },
//   patientId: {
//     type: DataTypes.STRING,
//   },
//   doctorId: {
//     type: DataTypes.STRING,
//   },
//   Htype: {
//     type: DataTypes.STRING,

//   },
//   hchiefComplaint: {
//     type: DataTypes.STRING,
//   },
//   height: {
//     type: DataTypes.NUMBER,
//   },
//   weight: {
//     type: DataTypes.NUMBER,
//   },
//   bmi: {
//     type: DataTypes.DECIMAL(10,2),
//   },
//   bloodGroup: {
//     type: DataTypes.STRING,
//   },

```

```
// pulse: {
//     type: DataTypes.NUMBER,
// },
// Bp: {
//     type: DataTypes.JSON,
// },
// durationInf: {
//     type: DataTypes.JSON,
// },
// marriageAge: {
//     type: DataTypes.NUMBER,
// },
// comment: {
//     type: DataTypes.TEXT,
// },
// note: {
//     type: DataTypes.TEXT,
// },
// Lmp: {
//     type: DataTypes.DATEONLY,
// },
// mH: {
//     type: DataTypes.JSON,
// },
// mHComment:{
//     type:DataTypes.STRING(2000)
// },
// fHComment: {
//     type: DataTypes.STRING(2000),
// },
// medicalHistory: {
//     type: DataTypes.JSON,
// },
// medicalHComment: {
//     type: DataTypes.STRING(2000),
// },
// oHPara: {
//     type: DataTypes.INTEGER(2),
// },
// oHAbortion: {
//     type: DataTypes.INTEGER(2),
// },
// oHEcotopic: {
//     type: DataTypes.INTEGER(2),
```

```

// },
// oHLiveBirth: {
//     type: DataTypes.INTEGER(2),
// },
// oHPregnancy: {
//     type: DataTypes.JSON,
// },
// rPL: {
//     type: DataTypes.STRING(3),
// },
// oHistoryComment: {
//     type: DataTypes.STRING(2000),
// },
// pastInflInvestigation:{
//     type:DataTypes.JSON,
// },
// pIIComment:{
//     type:DataTypes.STRING(2000),
// },
// partnerheight: {
//     type: DataTypes.NUMBER,
// },
// partnerweight: {
//     type: DataTypes.NUMBER,
// },
// partnerbmi: {
//     type: DataTypes.DECIMAL(10,2),
// },
// partnerbloodGroup: {
//     type: DataTypes.STRING,
// },
// partnerpulse: {
//     type: DataTypes.NUMBER,
// },
// partnerBp: {
//     type: DataTypes.JSON,
// },
// partnerComment:{
//     type:DataTypes.STRING(2000),
// },
// partnerPIInflInv:{
//     type:DataTypes.JSON
// },
// otherRelH:{

```



```

//     type: DataTypes.STRING(2000),
//   },
//   Date: {
//     type: DataTypes.DATEONLY,
//   },
// }, {
//   tableName: 'emb_patienthistory',
//   alter: true,
//   timestamps: true // Enable this if you want createdAt and updatedAt timestamps
// });
const treatmentHist = sequelize.define(
  "treatmentHist",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
    treatHist: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    tableName: "emb_treatmenthist",
    alter: true,
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);

const GeneralExamination = sequelize.define(
  "GeneralExamination",
  {

```

```

clinic_id: {
  type: DataTypes.INTEGER,
  defaultValue: 0,
},
cycleId: {
  type: DataTypes.INTEGER,
  defaultValue: 0,
},
patientId: {
  type: DataTypes.STRING,
},
doctorId: {
  type: DataTypes.STRING,
},
GeneralExamination: {
  type: DataTypes.JSON,
},
UsgExamination: {
  type: DataTypes.JSON,
},

comments: {
  type: DataTypes.TEXT, // Maximum of 2000 characters
},
notesAdvice: {
  type: DataTypes.TEXT, // Maximum of 2000 characters
},
},
{
  timestamps: true,
  alter: true,
  tableName: "emb_generalexaminations",
}
);

```

```

const diagnosis = sequelize.define(
  "diagnosis",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,

```

```

    defaultValue: 0,
  },
  patientId: {
    type: DataTypes.STRING,
  },
  doctorId: {
    type: DataTypes.STRING,
  },
  infType: {
    type: DataTypes.STRING,
  },
  maleFactor: {
    type: DataTypes.STRING,
  },
  maleFactorRemarks: {
    type: DataTypes.STRING,
  },
  femaleFactor: {
    type: DataTypes.STRING,
  },
  femaleFactorRemarks: {
    type: DataTypes.STRING,
  },
  karyotypeAbnormality: {
    type: DataTypes.STRING,
  },
  karyotypeAbnormalityRemarks: {
    type: DataTypes.STRING,
  },
  noOfInfYear: {
    type: DataTypes.INTEGER(3),
  },
  other: {
    type: DataTypes.STRING,
  },
  Date: {
    type: DataTypes.DATEONLY,
  },
},
{
  tableName: "emb_diagnosis",
  alter: true,
  timestamps: true, // Enable this if you want createdAt and updatedAt timestamps

```

```
}  
);
```

```
const doctorNotes = sequelize.define(  
  "doctorNotes",  
  {  
    clinic_id: {  
      type: DataTypes.INTEGER,  
      defaultValue: 0,  
    },  
    cycleId: {  
      type: DataTypes.INTEGER,  
      defaultValue: 0,  
    },  
    patientId: {  
      type: DataTypes.STRING,  
    },  
    doctorId: {  
      type: DataTypes.STRING,  
    },  
    doctorNotes: {  
      type: DataTypes.JSON,  
    },  
    Date: {  
      type: DataTypes.DATEONLY,  
    },  
  },  
  {  
    tableName: "emb_doctornotes",  
    alter: true,  
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps  
  }  
);
```

```
const serviceAdvice = sequelize.define(  
  "serviceAdvice",  
  {  
    clinic_id: {  
      type: DataTypes.INTEGER,  
      defaultValue: 0,  
    },  
    cycleId: {  
      type: DataTypes.INTEGER,  
      defaultValue: 0,  
    },  
  },  
  {  
    tableName: "emb_serviceadvice",  
    alter: true,  
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps  
  }  
);
```

```

    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
    ServiceAdvice: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    tableName: "emb_serviceadvices",
    alter: true,
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);

```

```

const prescription = sequelize.define(
  "prescription",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
    prescribedMedicine: {
      type: DataTypes.JSON,
    },
    prescribedComment: {
      type: DataTypes.STRING,
    },
  },

```

```

    favourites: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    tableName: "emb_prescription",
    alter: true,
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);

```

```

const attachments = sequelize.define(
  "attachments",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
    attachment: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    {
      tableName: "emb_attachments",
      alter: true,
      timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
    }
  }
);

```

```

const FollowUp = sequelize.define(
  "FollowUp",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.STRING,
    },
    followUp: {
      type: DataTypes.JSON,
      // Assuming this field is optional
    },
    appointmentTime: {
      type: DataTypes.TIME,
    },
  },
  {
    tableName: "emb_followups",
    alter: true,
    timestamps: true, // Assuming you want to track creation and update timestamps
  }
);

```

```

// FollowUp.sync({ alter: true });

```

```

const procedureAdvice = sequelize.define(
  "procedureAdvice",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    cycleId: {
      type: DataTypes.INTEGER,

```

```

    defaultValue: 0,
  },
  patientId: {
    type: DataTypes.STRING,
  },
  doctorId: {
    type: DataTypes.STRING,
  },
  procedure: {
    type: DataTypes.JSON,
    // Assuming this field is optional
  },
},
{
  tableName: "emb_procedureadvice",
  alter: true,
  timestamps: true, // Assuming you want to track creation and update timestamps
}
);

```

```

const Day0Record = sequelize.define(
  "Day0Record",
  {
    cycleId: {
      type: DataTypes.INTEGER,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    Date: {
      type: DataTypes.DATEONLY, // Store the entire table data as JSON
    },
    Time: {
      type: DataTypes.TIME, // Store the entire table data as JSON
    },
    FertiMethod: {
      type: DataTypes.STRING,
    },
    primEmbriologist: {
      type: DataTypes.STRING,
    },
  },

```



```

    secEmbriologist: {
      type: DataTypes.STRING,
    },
    incubator: {
      type: DataTypes.STRING,
    },
    remarks: {
      type: DataTypes.STRING,
    },
    reportData: {
      type: DataTypes.JSON, // Store the entire table data as JSON
    },
  },
  {
    tableName: "emb_day0record", // Customize the table name if necessary
    alter: true,
    timestamps: true, // Adds createdAt and updatedAt fields automatically
  }
);

```

```

const Day1Record = sequelize.define(
  "Day1Record",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    primEmbriologist: {
      type: DataTypes.STRING,
    },
    secEmbriologist: {
      type: DataTypes.STRING,
    },
    Date: {
      type: DataTypes.DATEONLY, // Store the entire table data as JSON
    },
    incubator: {
      type: DataTypes.STRING,
    },
  },
  {
    timestamps: true,
  }
);

```

```

    },
    remarks: {
      type: DataTypes.STRING,
    },
    reportData: {
      type: DataTypes.JSON, // Store the entire table data as JSON
    },
  },
  {
    tableName: "emb_day1record", // Customize the table name if necessary
    alter: true,
    timestamps: true, // Adds createdAt and updatedAt fields automatically
  }
);

```

```

const Day2Record = sequelize.define(
  "Day2Record",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    Date: {
      type: DataTypes.DATEONLY, // Store the entire table data as JSON
    },
    primEmbriologist: {
      type: DataTypes.STRING,
    },
    secEmbriologist: {
      type: DataTypes.STRING,
    },
    incubator: {
      type: DataTypes.STRING,
    },
    remarks: {
      type: DataTypes.STRING,
    },
    reportData: {

```

```

    type: DataTypes.JSON, // Store the entire table data as JSON
  },
},
{
  tableName: "emb_day2record", // Customize the table name if necessary
  alter: true,
  timestamps: true, // Adds createdAt and updatedAt fields automatically
}
);
const Day3Record = sequelize.define(
  "Day3Record",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    Date: {
      type: DataTypes.DATEONLY, // Store the entire table data as JSON
    },
    primEmbriologist: {
      type: DataTypes.STRING,
    },
    secEmbriologist: {
      type: DataTypes.STRING,
    },
    incubator: {
      type: DataTypes.STRING,
    },
    remarks: {
      type: DataTypes.STRING,
    },
    reportData: {
      type: DataTypes.JSON, // Store the entire table data as JSON
    },
  },
  {
    {
      tableName: "emb_day3record", // Customize the table name if necessary
      alter: true,

```

```

    timestamps: true, // Adds createdAt and updatedAt fields automatically
  }
);

const Day4Record = sequelize.define(
  "Day4Record",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    Date: {
      type: DataTypes.DATEONLY, // Store the entire table data as JSON
    },
    primEmbriologist: {
      type: DataTypes.STRING,
    },
    secEmbriologist: {
      type: DataTypes.STRING,
    },
    incubator: {
      type: DataTypes.STRING,
    },
    remarks: {
      type: DataTypes.STRING,
    },
    reportData: {
      type: DataTypes.JSON, // Store the entire table data as JSON
    },
  },
  {
    tableName: "emb_day4record", // Customize the table name if necessary
    alter: true,
    timestamps: true, // Adds createdAt and updatedAt fields automatically
  }
);

const Day5Record = sequelize.define(
  "Day5Record",

```

```

{
  cycleId: {
    type: DataTypes.INTEGER,
    defaultValue: 0,
  },
  patientId: {
    type: DataTypes.INTEGER,
  },
  clinicId: {
    type: DataTypes.INTEGER,
  },
  Date: {
    type: DataTypes.DATEONLY, // Store the entire table data as JSON
  },
  primEmbriologist: {
    type: DataTypes.STRING,
  },
  secEmbriologist: {
    type: DataTypes.STRING,
  },
  incubator: {
    type: DataTypes.STRING,
  },
  remarks: {
    type: DataTypes.STRING,
  },
  reportData: {
    type: DataTypes.JSON, // Store the entire table data as JSON
  },
},
{
  tableName: "emb_day5record", // Customize the table name if necessary
  alter: true,
  timestamps: true, // Adds createdAt and updatedAt fields automatically
}
);

const Day6Record = sequelize.define(
  "Day6Record",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {

```

```

    type: DataTypes.INTEGER,
  },
  clinicId: {
    type: DataTypes.INTEGER,
  },
  primEmbriologist: {
    type: DataTypes.STRING,
  },
  Date: {
    type: DataTypes.DATEONLY, // Store the entire table data as JSON
  },
  secEmbriologist: {
    type: DataTypes.STRING,
  },
  incubator: {
    type: DataTypes.STRING,
  },
  remarks: {
    type: DataTypes.STRING,
  },
  reportData: {
    type: DataTypes.JSON, // Store the entire table data as JSON
  },
},
{
  tableName: "emb_day6record", // Customize the table name if necessary
  alter: true,
  timestamps: true, // Adds createdAt and updatedAt fields automatically
}
);

```

```

const SpermAdvanceResult = sequelize.define(
  "SpermAdvanceResult",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    referenceDate: {
      type: DataTypes.STRING,
    },
    referenceNo: {
      type: DataTypes.STRING,
    },
  },

```

```
clinic: {
  type: DataTypes.STRING,
},
uhid: {
  type: DataTypes.STRING,
},
patientAccount: {
  type: DataTypes.STRING,
},
patientRegistration: {
  type: DataTypes.STRING,
},
investigationRequestNo: {
  type: DataTypes.STRING,
},
investigationType: {
  type: DataTypes.STRING,
},
sampleReceivingDate: {
  type: DataTypes.DATEONLY,
},
sampleReceivingTime: {
  type: DataTypes.TIME,
},
sampleTakenDate: {
  type: DataTypes.DATEONLY,
},
sampleTakenTime: {
  type: DataTypes.TIME,
},
spermCollectionType: {
  type: DataTypes.STRING,
},
collectionAt: {
  type: DataTypes.STRING,
},
preparedBy: {
  type: DataTypes.STRING,
},
checkedBy: {
  type: DataTypes.STRING,
},
remark: {
  type: DataTypes.STRING,
```

```
,
anySpillageReported: {
  type: DataTypes.STRING,
},
volume: {
  type: DataTypes.STRING,
},
visualAppearance: {
  type: DataTypes.STRING,
},
examinationInitiatedTime: {
  type: DataTypes.STRING,
},
examinationEndTime: {
  type: DataTypes.STRING,
},
duration: {
  type: DataTypes.STRING,
},
liquefaction: {
  type: DataTypes.STRING,
},
liquefactionTime: {
  type: DataTypes.STRING,
},
ph: {
  type: DataTypes.STRING,
},
viscosity: {
  type: DataTypes.STRING,
},
abstinencePeriod: {
  type: DataTypes.STRING,
},
spermConcentration: {
  type: DataTypes.STRING,
},
totalSpermNumber: {
  type: DataTypes.STRING,
},
rapidProgressive: {
  type: DataTypes.STRING,
},
slowProgressive: {
```



```
    type: DataTypes.STRING,
  },
  prProgressive: {
    type: DataTypes.STRING,
  },
  nonProgressive: {
    type: DataTypes.STRING,
  },
  totalMotile: {
    type: DataTypes.STRING,
  },
  immotile: {
    type: DataTypes.STRING,
  },
  debrisName: {
    type: DataTypes.STRING,
  },
  roundCells: {
    type: DataTypes.STRING,
  },
  agglutinationName: {
    type: DataTypes.STRING,
  },
  aggregationName: {
    type: DataTypes.STRING,
  },
  enterViability: {
    type: DataTypes.STRING,
  },
  spermViability: {
    type: DataTypes.STRING,
  },
  fructoseName: {
    type: DataTypes.STRING,
  },
  peroxidasePositiveCollisConcentration: {
    type: DataTypes.STRING,
  },
  systemImpression: {
    type: DataTypes.STRING,
  },
  impression: {
    type: DataTypes.STRING,
  },
}
```

```

    },
    {
      tableName: "emb_spermadvanceresult", // Customize the table name if necessary
      alter: true,
      timestamps: true, // Adds createdAt and updatedAt fields automatically
    }
  );

```

```

const stimulation = sequelize.define(
  "stimulation",
  {
    cycleId: {
      type: DataTypes.INTEGER,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    DonorId: {
      type: DataTypes.STRING,
    },
    dayId: {
      type: DataTypes.INTEGER,
    },
    option: {
      type: DataTypes.STRING,
    },
    StimulationRows: {
      type: DataTypes.JSON,
      // Assuming this field is optional
    },
  },
  {
    {
      tableName: "emb_stimulation",
      alter: true,
      timestamps: true, // Assuming you want to track creation and update timestamps
    }
  }
);

```

```

const StimulationSubmit = sequelize.define(
  "StimulationSubmit",
  {

```

```
cycleId: {
  type: DataTypes.INTEGER,
},
patientId: {
  type: DataTypes.INTEGER,
},
clinicId: {
  type: DataTypes.INTEGER,
},
DonorId: {
  type: DataTypes.STRING,
},
stimulationCompletedOn: {
  type: DataTypes.DATEONLY,
},
readyForOpu: {
  type: DataTypes.BOOLEAN,
},
failedCycle: {
  type: DataTypes.BOOLEAN,
},
reasonForFailure: {
  type: DataTypes.STRING,
},
stimulationRecordedBy: {
  type: DataTypes.STRING,
},
transfer: {
  type: DataTypes.STRING,
},
remarks: {
  type: DataTypes.TEXT,
},
firstTrigger: {
  type: DataTypes.JSON,
},
triggerType: {
  type: DataTypes.STRING,
},
opuOn: {
  type: DataTypes.STRING,
},
noOfFollicles: {
  type: DataTypes.INTEGER,
```

```

    },
    drug: {
      type: DataTypes.JSON,
    },
  },
  {
    tableName: "emb_stimulationsubmit",
    alter: true,
    timestamps: true,
  }
);

```

```

const ScheduleEt = sequelize.define(
  "ScheduleEt",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    etDate: {
      type: DataTypes.DATEONLY,
    },
    etTime: {
      type: DataTypes.TIME,
    },
    etTimeERA: {
      type: DataTypes.STRING,
      defaultValue: false,
    },
    reviewedBy: {
      type: DataTypes.STRING,
    },
    fertStatus: {
      type: DataTypes.STRING,
      defaultValue: false,
    },
  },
  {
    {

```

```

    tableName: "emb_schedule_et",
    alter: true,
    timestamps: true, // createdAt and updatedAt fields are automatically added
  }
);

const BetaHcg = sequelize.define(
  "BetaHcg",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
      allowNull: false, // Ensures patientId is required
    },
    clinicId: {
      type: DataTypes.INTEGER,
      allowNull: false, // Ensures clinicId is required
    },
    betahcgValue: {
      type: DataTypes.JSON,
      allowNull: true,
    },
    betaHcgDate: {
      type: DataTypes.DATEONLY, // Store date only (without time)
      allowNull: true,
    },
    recordedBy: {
      type: DataTypes.STRING, // You can use STRING to store the doctor's name or ID
      allowNull: true,
    },
    observations: {
      type: DataTypes.TEXT, // To store free-form observations
      allowNull: true,
    },
  },
  {
    tableName: "emb_beta_hcg_records", // The table name in your database
    timestamps: true, // createdAt and updatedAt fields are automatically added
    alter: true, // Allows Sequelize to update the schema if needed
  }
);

```

```
const Outcome = sequelize.define(
  "Outcome",
  {
    cycleId: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    clinicalPregnancyDate: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    clinicalPregnancyRecordedBy: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    clinicalPregnancyLocation: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    noOfSac: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    clinicalPregnancyObservations: {
      type: DataTypes.TEXT,
      allowNull: true,
    },
    ongoingPregnancyObservations: {
      type: DataTypes.TEXT,
      allowNull: true,
    },
    miscarriageDate: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    miscarriageTime: {
      type: DataTypes.STRING,
```

```

    allowNull: true,
  },
  abortionType: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  clinicalMiscarriageObservations: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  deliveryDate2: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  birthRecordedBy: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  birthType: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  modeOfDelivery: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  neonatalComplications: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  postPartumMaternalComplications: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  birthDetailsObservations: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  biochemicalPregnancy: {
    type: DataTypes.STRING,
  },
},
{
  tableName: "emb_outcome_records",

```

```
    alter: true,  
    timestamps: true, // createdAt and updatedAt fields are automatically added  
  }  
);
```

```
// Outcome.sync({ alter: true });
```

```
const ovumSchedule = sequelize.define(  
  "ovumSchema",  
  {  
    cycleId: {  
      type: DataTypes.INTEGER,  
    },  
    DonorId: {  
      type: DataTypes.STRING,  
    },  
    clinchId: {  
      type: DataTypes.INTEGER,  
      defaultValue: 1,  
    },  
    patient_id: {  
      type: DataTypes.INTEGER,  
    },  
    Date: {  
      type: DataTypes.DATE,  
    },  
    Time: {  
      type: DataTypes.TIME,  
    },  
    Surgeon: {  
      type: DataTypes.STRING,  
    },  
    AssistantSurgeon: {  
      type: DataTypes.STRING,  
    },  
    Anesthetist: {  
      type: DataTypes.STRING,  
    },  
    AnesthesiaType: {  
      type: DataTypes.STRING,  
    },  
    PrimaryEmbryologist: {  
      type: DataTypes.STRING,  
    },  
  },
```



```

    SecondaryEmbryologist: {
      type: DataTypes.STRING,
    },
    OPUNotes: {
      type: DataTypes.STRING,
    },
    NoOfRightFollicles: {
      type: DataTypes.INTEGER,
    },
    NoOfLeftFollicles: {
      type: DataTypes.INTEGER,
    },
    NoOfOocytes: {
      type: DataTypes.INTEGER,
    },
    DenudationDate: {
      type: DataTypes.DATE,
    },
    DenudationTime: {
      type: DataTypes.TIME,
    },
    DenudationDoneBy: {
      type: DataTypes.STRING,
    },
    Option: {
      type: DataTypes.JSON,
    },
    Vitrification: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
  },
  {
    tableName: "embryo_ovumschedule",
    defaultValue: "false",
  }
);

// MINE
//

const PatientCounseling = sequelize.define(
  "PatientCounseling",
  {

```

```
cycle_id: {
  type: DataTypes.INTEGER,
  primaryKey: true,
  autoIncrement: true,
},
donorId: {
  type: DataTypes.STRING(50),
  allowNull: true,
},
patientId: {
  type: DataTypes.INTEGER,
},
clinicId: {
  type: DataTypes.INTEGER,
},
mrn: {
  type: DataTypes.STRING,
},
package_id: {
  type: DataTypes.INTEGER,
},
treatmentType: {
  type: DataTypes.STRING,
  allowNull: true,
},
assignedDoctor: {
  type: DataTypes.STRING,
  allowNull: true,
},
assignedAssistantDoctor: {
  type: DataTypes.STRING,
  allowNull: true,
},
assignedCounselor: {
  type: DataTypes.STRING,
  allowNull: true,
},
assignedCoordinator: {
  type: DataTypes.STRING,
  allowNull: true,
},
clinicalCounselingDoneBy: {
  type: DataTypes.STRING,
  allowNull: true,
}
```

```
},
clinicalCounselingDate: {
  type: DataTypes.DATE,
  allowNull: true,
},
financialCounselingDoneBy: {
  type: DataTypes.STRING,
  allowNull: true,
},
financialCounselingDate: {
  type: DataTypes.DATE,
  allowNull: true,
},
notesDate: {
  type: DataTypes.DATE,
  allowNull: true,
},
notes: {
  type: DataTypes.TEXT,
  allowNull: true,
},
notesRecordedBy: {
  type: DataTypes.STRING,
  allowNull: true,
},
amount: {
  type: DataTypes.JSON,
},
status: {
  type: DataTypes.BOOLEAN,
  defaultValue: true,
},
currentPage: {
  type: DataTypes.STRING,
},
spermDonor: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
oocyteDonor: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},
frozenOocyte: {
```

```

    type: DataTypes.BOOLEAN,
    defaultValue: false,
  },
},
{
  timestamps: true,
  tableName: "patient_counseling",
}
);

// PatientCounseling.sync({ force: true });

// geetapage.js

const History = sequelize.define(
  "History",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    doctorId: {
      type: DataTypes.INTEGER,
    },
    History: {
      type: DataTypes.JSON,
    },
    Date: {
      type: DataTypes.DATEONLY,
    },
  },
  {
    tableName: "emb_history",
    alter: true,
    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);
// History.sync({ force: true });

const Package = sequelize.define(
  "Package",

```

```
{
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  packageCode: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  packageName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  packageRate: {
    type: DataTypes.INTEGER,
    allowNull: true,
  },
  packageEffDate: {
    type: DataTypes.DATE,
    allowNull: true,
  },
  packageExpDate: {
    type: DataTypes.DATE,
    allowNull: true,
  },
  specialization: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  subSpecialization: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  shortDescription: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  longDescription: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  service_tax: {
    type: DataTypes.STRING,
```

```
    allowNull: true,
  },
  service_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  service_amount: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  staffDiscount: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  staffDiscount_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  staffDiscount_amount: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  staffParentAccount: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  staffParent_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  staffParent_amount: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  concession: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  concession_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  concession_amount: {
    type: DataTypes.STRING,
```

```
    defaultValue: "0",
  },
  doctor: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  doctor_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  doctor_amount: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  rateEditable: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  rateEditable_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  rateEditable_amount: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  S_Citizen_Con: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  S_Citizen_per: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  S_Citizen_amount: {
    type: DataTypes.STRING,
    defaultValue: "0",
  },
  family: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  count: {
    type: DataTypes.INTEGER,
```

```

    defaultValue: 0,
  },
  selectAll: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  applicableMemberRelations: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: DataTypes.NOW,
  },
  updatedAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: DataTypes.NOW,
  },
},
{
  tableName: "pkg_newpackage",
  timestamps: true, // This will handle `createdAt` and `updatedAt` automatically
}
);

```

```

const ItemMasterNew = sequelize.define(
  "ItemMaster",
  {
    item_code: {
      type: DataTypes.STRING,
    },
    brand_name: {
      type: DataTypes.STRING,
    },
    item_name: {
      type: DataTypes.STRING,
    },
    item_group: {
      type: DataTypes.STRING,
    },
    dispensing_type: {
      type: DataTypes.STRING,
    },
  },
  {
    timestamps: true,
  },
);

```



```
    allowNull: true,
  },
  pregnancy_class: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  manufactured_by: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  purchase_uom: {
    type: DataTypes.STRING,
  },
  base_uom: {
    type: DataTypes.STRING,
  },
  base_unit_mrp: {
    type: DataTypes.FLOAT,
  },
  discount_on_sale: {
    type: DataTypes.FLOAT,
  },
  AnalysisRequired: {
    type: DataTypes.STRING,
  },
  expiry_alert_before_in_days: {
    type: DataTypes.INTEGER,
  },
  hsn_codes: {
    type: DataTypes.STRING,
  },
  walk_in_patients_discount_on_sale: {
    type: DataTypes.FLOAT,
  },
  staff_discount_on_sale: {
    type: DataTypes.FLOAT,
  },
  registered_patients_discount_on_sale: {
    type: DataTypes.FLOAT,
  },
  strength: {
    type: DataTypes.STRING,
    allowNull: true,
  },
}
```

```
strength_unit: {
  type: DataTypes.STRING,
  allowNull: true,
},
molecule_name: {
  type: DataTypes.STRING,
  allowNull: true,
},
item_category: {
  type: DataTypes.STRING,
},
storage_type: {
  type: DataTypes.STRING,
},
storage_degree: {
  type: DataTypes.FLOAT,
},
therapeutic_class: {
  type: DataTypes.STRING,
  allowNull: true,
},
stocking_uom: {
  type: DataTypes.STRING,
},
selling_uom: {
  type: DataTypes.STRING,
},
base_unit_cost_price: {
  type: DataTypes.FLOAT,
},
route: {
  type: DataTypes.STRING,
},
batchesRequired: {
  type: DataTypes.STRING,
},
batchesRequired: {
  type: DataTypes.STRING,
},
suspend: {
  type: DataTypes.BOOLEAN,
},
},
{
```

```

    alert: true,
    timestamps: true,
    tableName: "adminvitemmaster",
  }
);

// ItemMasterNew.sync({ alter: true });
const CurrentItemStock = sequelize.define(
  "InventoryItem",
  {
    clinic: {
      type: DataTypes.STRING,
    },
    store: {
      type: DataTypes.STRING,
    },

    item_id: {
      type: DataTypes.INTEGER,
    },
    isFree: {
      type: DataTypes.BOOLEAN,
    },

    batchCode: {
      type: DataTypes.STRING,
    },
    availableStock: {
      type: DataTypes.INTEGER,
    },

    expiryDate: {
      type: DataTypes.DATE,
    },
  },
  {
    tableName: "invcurrentitemstock",
    timestamps: true, // You can change this if you need timestamps
  }
);

// consent

const PatientConsent = sequelize.define(

```

```

"PatientConsent",
{
  patient_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
  form_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },

  is_filled: {
    type: DataTypes.BOOLEAN,
    defaultValue: false,
  },
  filled_at: {
    type: DataTypes.DATE,
    allowNull: true,
  },
},
{
  tableName: "patient_consents",
  timestamps: true,
  createdAt: "created_at",
  updatedAt: "updated_at",
}
);
// consent

// CanisterData.sync();

// Sync all models
// sequelize
// .sync({ force: true })
// .then(() => {
//   console.log("All tables have been recreated successfully.");
// })
// .catch((error) => {
//   console.error("Error recreating tables:", error);
// });

// PatientCounseling.sync({ alter: true }) // Use { force: true } if you want to drop and recreate the
table
// .then(() => {

```

```

// console.log("Donor table synced successfully.");
// })
// .catch((error) => {
// console.error("Error syncing Donor table:", error);
// });
const NotificationTracking = sequelize.define(
  "NotificationTracking",
  {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
      allowNull: false,
    },
    patientId: {
      type: DataTypes.STRING,
    },
    user_id: {
      type: DataTypes.INTEGER,
      allowNull: true, // Set to true for expiring items since no user_id is needed here
    },
    kyc_id: {
      type: DataTypes.INTEGER,
      allowNull: true, // Set to true for expiring items since no kyc_id is needed here
    },
    apt_date: {
      type: DataTypes.DATEONLY,
      allowNull: true, // Set to true for expiring items
    },
    item_code: {
      type: DataTypes.STRING,
      allowNull: true, // Field for tracking expiring items
    },
    batchcode: {
      type: DataTypes.STRING,
      allowNull: true, // Field for tracking batch code from CurrentItemStock
    },
    notification_type: {
      type: DataTypes.STRING,
      allowNull: false, // New field to differentiate types of notifications
      defaultValue: "appointment", // Can be 'appointment' or 'expiry' or desktop
    },
    notification_status: {
      type: DataTypes.BOOLEAN,

```

```

    defaultValue: false,
  },
  notification_date: {
    type: DataTypes.DATE,
    defaultValue: DataTypes.NOW,
  },
},
{
  timestamps: false, // Disable timestamps if not needed
  tableName: "notification_tracking",
}
);

```

```

const vitrifyOocyte = sequelize.define(
  "vitrifyOocyte",
  {
    cycleId: {
      type: DataTypes.INTEGER,
    },
    oldCycleId: {
      type: DataTypes.INTEGER,
    },
    patientId: {
      type: DataTypes.INTEGER,
    },
    clinicId: {
      type: DataTypes.INTEGER,
    },
    vitrificationDate: {
      type: DataTypes.DATEONLY,
    },
    oocyteNumber: {
      type: DataTypes.STRING,
    },
    incubator: {
      type: DataTypes.STRING,
    },
    vitrificationTime: {
      type: DataTypes.TIME,
    },
    tank: {
      type: DataTypes.STRING,
    },
    canister: {

```

```

    type: DataTypes.STRING,
  },
  goble: {
    type: DataTypes.STRING,
  },
  hexatube: {
    type: DataTypes.STRING,
  },
  visotube: {
    type: DataTypes.STRING,
  },
  cryotopColorAndNumber: {
    type: DataTypes.STRING,
  },
  doneBy: {
    type: DataTypes.STRING,
  },
  witnessedBy: {
    type: DataTypes.STRING,
  },
  status: {
    type: DataTypes.ENUM("freeze", "observe", "consumed"),
    defaultValue: "freeze",
  },
},
{
  timestamps: false, // Disable timestamps if not needed
  tableName: "emb_vitrifyoocytes",
}
);

```

```

// <<<<<<< Updated upstream
// vitrifyOocyte.sync({ force: true });
// NotificationTracking.sync({ alter: true });
// =====

```

```

const vitrifyEmbryos = sequelize.define(
  "VitrifyEmbryos",
  {
    cycleId: {
      type: DataTypes.INTEGER,
    },
    parent_cycleId: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
  },

```

```
},
patientId: {
  type: DataTypes.INTEGER,
  allowNull: false,
},
clinicId: {
  type: DataTypes.INTEGER,
  allowNull: false,
},
DayEmbryo: {
  type: DataTypes.STRING,
  allowNull: true,
},
vitrificationDate: {
  type: DataTypes.DATEONLY,
  allowNull: true,
},
oocyteNumber: {
  type: DataTypes.STRING,
  allowNull: true,
},
incubator: {
  type: DataTypes.STRING,
  allowNull: true,
},
vitrificationTime: {
  type: DataTypes.TIME,
  allowNull: true,
},
tank: {
  type: DataTypes.STRING,
  allowNull: true,
},
canister: {
  type: DataTypes.STRING,
  allowNull: true,
},
goble: {
  type: DataTypes.STRING,
  allowNull: true,
},
hexatube: {
  type: DataTypes.STRING,
  allowNull: true,
```



```

    },
    visotube: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    cryotopColorAndNumber: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    doneBy: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    witnessedBy: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    status: {
      type: DataTypes.ENUM("freeze", "observe", "consumed"),
      defaultValue: "freeze",
    },
  },
  {
    timestamps: false, // Disable timestamps if not needed
    tableName: "emb_vitrifyembryos",
  }
);

```

```
// PatientCounseling.sync({alter: true});
```

```
// vitrifyOocyte.sync({ force: true });
```

```
// sequelize.sync({ alter: true });
```

```
module.exports = {
```

```
  BetaHcg,
```

```
  Donor,
```

```
  DonorOocyte,
```

```
  DonorVisit,
```

```
  SemenSample,
```

```
  vitrifyOocyte,
```

```
  vitrifyEmbryos,
```

```
  WashSemenSample,
```

```
  DiscardedSample,
```

```
  EmbryoTransfer,
```

```
  CanisterData,
```

```
  treatmentAdvice,
```

```
chiefComplaint,  
preExistingCondition,  
allergies,  
treatmentHist,  
GeneralExamination,  
diagnosis,  
doctorNotes,  
serviceAdvice,  
prescription,  
attachments,  
FollowUp,  
procedureAdvice,  
Day0Record,  
Day1Record,  
Day2Record,  
Day3Record,  
Day4Record,  
Day5Record,  
Day6Record,  
SpermAdvanceResult,  
stimulation,  
StimulationSubmit,  
ScheduleEt,  
Outcome,  
ovumSchedule,  
PatientCounseling,  
History,  
Package,  
ItemMasterNew,  
CurrentItemStock,  
NotificationTracking,  
};
```

PATIENT :-

```
const PR_patientReg = sequelize.define(  
  "PR_patientReg",  
  {  
    id: {  
      type: DataTypes.INTEGER,  
      autoIncrement: true,  
      primaryKey: true,  
    },  
  },
```

```
clinic_id: {
  type: DataTypes.INTEGER,
  defaultValue: 0,
},
mr_no: {
  type: DataTypes.STRING(255),
},
ArtBankName: {
  type: DataTypes.TEXT,
},
Agent: {
  type: DataTypes.TEXT,
},
patientImg: {
  type: DataTypes.STRING,
},
spouseImg: {
  type: DataTypes.STRING,
},
ReferralDetails: {
  type: DataTypes.TEXT,
},
RegistrationType: {
  type: DataTypes.TEXT,
},
SorOfRef: {
  type: DataTypes.TEXT,
},
address: {
  type: DataTypes.TEXT,
},
age_days: {
  type: DataTypes.STRING,
  defaultValue: "",
},
age_months: {
  type: DataTypes.STRING,
  defaultValue: "",
},
age_years: {
  type: DataTypes.STRING,
  defaultValue: "",
},
altMobileNo: {
```

```
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  area: {
    type: DataTypes.TEXT,
  },
  bloodGroup: {
    type: DataTypes.STRING(10),
    defaultValue: "",
  },
  camp: {
    type: DataTypes.TEXT,
  },
  city: {
    type: DataTypes.TEXT,
  },
  clinic_status: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  company_name: {
    type: DataTypes.TEXT,
  },
  country: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  date: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  Gender: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  dob: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  education: {
    type: DataTypes.TEXT,
  },
  email: {
    type: DataTypes.STRING(100),
```

```
    defaultValue: "",
  },
  familyName: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  fatherName: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  firstName: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  id_proof_number: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  id_proof_type: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  inHouseDoctor: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  is_vip: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  is_employee: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  is_insured: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  is_international: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  externalDoctor: {
    type: DataTypes.STRING(100),
```

```
    defaultValue: "",
  },
  landlineNo: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  lastName: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  marital_status: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  marriage_anniversary: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  middleName: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  mobileNo: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  mobile_1: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  mobile_2: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  occupation: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  phone1: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  phone2: {
    type: DataTypes.STRING(100),
```

```
    defaultValue: "",
  },
  pin_code: {
    type: DataTypes.STRING(20),
    defaultValue: "",
  },
  prefix: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  religion: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },

  special_registration: {
    type: DataTypes.STRING(50),
    defaultValue: "",
  },
  state: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  spouse_address: {
    type: DataTypes.TEXT,
  },
  spouse_age_day: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  spouse_age_month: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  spouse_age_year: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  spouse_area: {
    type: DataTypes.TEXT,
  },
  spouse_blood_group: {
    type: DataTypes.STRING(10),
    defaultValue: "",
  },
```

```
},
spouse_city: {
  type: DataTypes.TEXT,
},
spouse_company_name: {
  type: DataTypes.TEXT,
},
spouse_country: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_dob: {
  type: DataTypes.STRING,
  defaultValue: "",
},
spouse_education: {
  type: DataTypes.TEXT,
},
spouse_email: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_familyName: {
  type: DataTypes.TEXT,
},
spouse_firstName: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_gender: {
  type: DataTypes.STRING(50),
  defaultValue: "",
},
spouse_id_proof: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_id_proof_number: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_is_international: {
  type: DataTypes.STRING(50),
  defaultValue: "",
}
```



```
},
spouse_lastName: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_marital_status: {
  type: DataTypes.STRING(50),
  defaultValue: "",
},
spouse_marriage_anniversary: {
  type: DataTypes.STRING,
},
spouse_middleName: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_mobile_1: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_mobile_2: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_motherName: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_nationality: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_occupation: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
spouse_pin_code: {
  type: DataTypes.STRING(20),
  defaultValue: "",
},
spouse_preferred_language: {
  type: DataTypes.STRING(50),
  defaultValue: "",
},
},
```

```
spouse_prefix: {  
  type: DataTypes.STRING(50),  
  defaultValue: "",  
},  
spouse_religion: {  
  type: DataTypes.STRING(50),  
  defaultValue: "",  
},  
spouse_residence_phone_1: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
spouse_residence_phone_2: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
spouse_residence_phone_3: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
spouse_special_registration: {  
  type: DataTypes.STRING(50),  
  defaultValue: "",  
},  
spouse_state: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
sponsor_associated_company: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
sponsor_company: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
sponsor_investigation_no: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},  
sponsor_member_relation: {  
  type: DataTypes.STRING(100),  
  defaultValue: "",  
},
```

```
sponsor_patient_category: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
sponsor_patient_source: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
sponsor_remark: {
  type: DataTypes.TEXT,
},
sponsor_tariff: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
ifsc_code: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
branch: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
bank_name: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
account_type: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
account_no: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
account_holder: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
height: {
  type: DataTypes.STRING(100),
  defaultValue: "",
},
weight: {
```

```

    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  bmi: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  coupleId: {
    type: DataTypes.STRING(100),
    defaultValue: "",
  },
  image_path: {
    type: DataTypes.STRING,
    allowNull: true, // Or false, depending on your requirements
  },
  spouse_path: {
    type: DataTypes.STRING,
    allowNull: true, // If you're saving spouse image too
  },
  dr_cabin: {
    type: DataTypes.STRING,
  },
  reasonOfVisit: {
    type: DataTypes.STRING,
  },
  department: {
    type: DataTypes.STRING,
  },
  doctor: {
    type: DataTypes.STRING,
  },
  remark: {
    type: DataTypes.TEXT,
  },
  referenceDoctor: {
    type: DataTypes.STRING,
  },
  visitNotes: {
    type: DataTypes.TEXT,
  },
},
{
  timestamps: true, // Set to true if you have createdAt and updatedAt columns
  alter: true,

```

```

    tableName: "pr_patientreg", // Ensure the table name is as per your database
  }
);
// PR_patientReg.sync({alter:true})
const PR_ReferralDoc = sequelize.define(
  "PR_ReferralDoc",
  {
    firstName: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    middleName: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    lastName: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    specialization: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    doctorType: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    gender: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    contactNumber: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    emailId: {
      type: DataTypes.STRING,
      defaultValue: "",
    },
    address: {
      type: DataTypes.TEXT,
      defaultValue: "",
    },
    PanCard: {

```

```

    type: DataTypes.STRING,
    defaultValue: "",
  },
  id_proof_type: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  id_proof_number: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  bank_name: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  branch: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  ifsc_code: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  account_no: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  account_holder: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
},
{
  timestamps: true,
  alter: true,
  tableName: "pr_referraldoc",
}
);

const PR_Appointment = sequelize.define(
  "PR_Appointment",
  {
    mrNo: {
      type: DataTypes.STRING,

```

```
},
age_days: {
  type: DataTypes.STRING,
  defaultValue: 0,
},
age_months: {
  type: DataTypes.STRING,
  defaultValue: 0,
},
age_years: {
  type: DataTypes.STRING,
  defaultValue: 0,
},
appointment_date: {
  type: DataTypes.STRING,
  defaultValue: "",
},
appointment_reason: {
  type: DataTypes.STRING,
  defaultValue: "",
},
clinic: {
  type: DataTypes.STRING,
  defaultValue: "",
},
department: {
  type: DataTypes.STRING,
  defaultValue: "",
},
dob: {
  type: DataTypes.STRING,
},
doctor: {
  type: DataTypes.STRING,
  defaultValue: "",
},
firstName: {
  type: DataTypes.STRING,
  defaultValue: "",
},
from_time: {
  type: DataTypes.STRING,
},
gender: {
```

```

    type: DataTypes.STRING,
  },
  lastName: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  mobile_code: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  mobile_number: {
    type: DataTypes.STRING,
    defaultValue: "",
  },
  remark: {
    type: DataTypes.STRING,
  },
  address: {
    type: DataTypes.STRING,
  },

  special_registration: {
    type: DataTypes.STRING,
  },
  to_time: {
    type: DataTypes.STRING,
  },
},
{
  tableName: "pr_appointment",
  alter: true,
  timestamps: true,
}
);

```

```

const PR_PatientVisit = sequelize.define(
  "PR_PatientVisit",
  {
    mrNo: {
      type: DataTypes.STRING,
    },
    unit: {
      type: DataTypes.STRING,
    },
  },

```



```

Visitdate: {
  type: DataTypes.DATEONLY,
},
time: {
  type: DataTypes.TIME,
},
reasonOfVisit: {
  type: DataTypes.STRING,
},
department: {
  type: DataTypes.STRING,
},
doctor: {
  type: DataTypes.STRING,
},
cabin: {
  type: DataTypes.STRING,
},
referenceDoctor: {
  type: DataTypes.STRING,
},
remarks: {
  type: DataTypes.TEXT,
},
visitNotes: {
  type: DataTypes.TEXT,
},
},
{
  tableName: "pr_patientvisits",
  alter: true,
  timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
}
);

```

```

const Patients = sequelize.define(
  "Patients",
  { clinic_id: {
    type: DataTypes.INTEGER, // Assuming age is an integer
  },
  PatientId: {
    type: DataTypes.INTEGER, // Assuming age is an integer
  },
  Name: {

```

```

    type: DataTypes.STRING,
  },
  Gender: {
    type: DataTypes.STRING, // Assuming gender is a string, can be 'Male', 'Female', etc.
  },
  MrNo: {
    type: DataTypes.STRING,
  },
  SpouseName: {
    type: DataTypes.STRING,
  },
  SpouseGender: {
    type: DataTypes.STRING, // Assuming gender of spouse
  },
  Age: {
    type: DataTypes.INTEGER, // Assuming age is an integer
  },
  Mobile: {
    type: DataTypes.STRING, // Assuming phone number is stored as a string
  },
  CoupleId: {
    type: DataTypes.INTEGER, // Assuming coupleId references another table
  },
  Dob: {
    type: DataTypes.DATEONLY, // Date of Birth (Only Date, without time)
  },
},
{
  tableName: "patients",
  alter: true,
  timestamps: true, // Enable createdAt and updatedAt timestamps
}
);

```

```

// Patients.sync({force:true})

```

```

const PR_formNewCouple = sequelize.define(
  "PR_formNewCouple",
  {
    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    mrNo: {

```

```

    type: DataTypes.STRING,
  },
  PatientName: {
    type: DataTypes.STRING,
  },
  Couple_mrNo: {
    type: DataTypes.STRING,
  },
  CoupleName: {
    type: DataTypes.STRING,
  },
  CoupleDob: {
    type: DataTypes.DATEONLY,
  },
},
{
  tableName: "pr_formnewcouple",
  alter: true,
  timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
}
);

```

```

const PR_BillFindPatient = sequelize.define(
  "PR_BillFindPatient",
  {

```

```

    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    bill_no: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    mrNo: {
      type: DataTypes.STRING,
    },
    selectedBillServices: {
      type: DataTypes.JSON,
    },
    date: {
      type: DataTypes.DATEONLY,
    },
    totalPayAmount: {

```

```
    type: DataTypes.STRING,
  },
  clinicBillAmount: {
    type: DataTypes.STRING,
  },
  totalNetAmount: {
    type: DataTypes.STRING,
  },
  totalConcessionAmount: {
    type: DataTypes.STRING,
  },
  totalNetBillAmount: {
    type: DataTypes.STRING,
  },
  totalBillAmount: {
    type: DataTypes.STRING,
  },
  doctor: {
    type: DataTypes.STRING,
  },
  remarks: {
    type: DataTypes.STRING,
  },
  Advance: {
    type: DataTypes.STRING,
  },
  ConcessionReason: {
    type: DataTypes.STRING,
  },
  freeze: {
    type: DataTypes.BOOLEAN,
    defaultValue: 0
  },

  approved: {
    type: DataTypes.BOOLEAN,
    defaultValue: false,
  },

  Approvalremark: {
    type: DataTypes.STRING,
  },
  approvedBy: {
    type: DataTypes.STRING,
```

```

    },
    discountRemarks: {
      type: DataTypes.TEXT,
    },
    billRemarks: {
      type: DataTypes.TEXT,
    },

    paidAmount: {
      type: DataTypes.STRING, // Assuming this is a numeric value stored as a string
    },
    balanceAmount: {
      type: DataTypes.STRING, // Assuming this is a numeric value stored as a string
    },
  },
  {
    tableName: "pr_billfindpatient",

    timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
  }
);
// PR_BillFindPatient.sync()

const PR_BillPharmacy = sequelize.define(
  "PR_BillPharmacy",
  {

    clinic_id: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    bill_no: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    mrNo: {
      type: DataTypes.STRING,
    },
    selectedBillItems: {
      type: DataTypes.JSON,
    },
    date: {
      type: DataTypes.DATEONLY,
    },
  },

```

```
totalPayAmount: {
  type: DataTypes.STRING,
},
pharmacyBillAmount: {
  type: DataTypes.STRING,
},
totalNetAmount: {
  type: DataTypes.STRING,
},
totalConcessionAmount: {
  type: DataTypes.STRING,
},
totalNetBillAmount: {
  type: DataTypes.STRING,
},
totalTaxableAmount: {
  type: DataTypes.STRING,
},
remarks: {
  type: DataTypes.STRING,
},
Advance: {
  type: DataTypes.STRING,
},
ConcessionReason: {
  type: DataTypes.STRING,
},
doctor: {
  type: DataTypes.STRING,
},
store: {
  type: DataTypes.STRING,
},
freeze: {
  type: DataTypes.BOOLEAN,
  defaultValue: 0
},

approved: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
},

Approvalremark: {
```

```

    type: DataTypes.STRING,
  },
  approvedBy: {
    type: DataTypes.STRING,
  },
  discountRemarks: {
    type: DataTypes.TEXT,
  },
  billRemarks: {
    type: DataTypes.TEXT,
  },

  paidAmount: {
    type: DataTypes.STRING, // Assuming this is a numeric value stored as a string
  },
  balanceAmount: {
    type: DataTypes.STRING, // Assuming this is a numeric value stored as a string
  },
},
{
  tableName: "pr_billpharmacy",
  alter: true,
  timestamps: true, // Enable this if you want createdAt and updatedAt timestamps
}
);
// PR_BillPharmacy.sync()

const PR_BillPaymentDetails = sequelize.define('PR_BillPaymentDetails', {
  bill_no: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  advancePayment: {
    type: DataTypes.STRING, // Assuming it is a numeric value stored as a string
    allowNull: false,
    defaultValue: '0.00',
  },
  adjustAdvancePayment: {
    type: DataTypes.STRING, // Assuming it is a numeric value stored as a string
    allowNull: true,
  },
  balanceAmount: {
    type: DataTypes.STRING, // Assuming it is a numeric value stored as a string
    allowNull: false,

```

```
    defaultValue: '0.00',
  },
  paymentMethod: {
    type: DataTypes.STRING, // Cash, Cheque, Card, TT, EPayment, NEFT, Insurance,
    Corporate
    allowNull: true,
  },
  recievedAmount: {
    type: DataTypes.STRING, // Assuming it is a numeric value stored as a string
    allowNull: true,
  },
  referenceNo: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  referenceDate: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  bankName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  bankDetails: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  defaultBankAccount: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  payer: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  refNo: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  validFrom: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  validTo: {
```



```
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  financialRemarks: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  company: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  insurance: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  refNoCorporate: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  validFromCorporate: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  validToCorporate: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  corporateRemarks: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  chequeNo: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  chequeDate: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  bankNameCheque: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  bankDetailsCheque: {
```

```
    type: DataTypes.TEXT,
    allowNull: true,
  },
  defaultBankAccountCheque: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  referenceNoEPayment: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  referenceDateEPayment: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  ePaymentGatewayName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  ePaymentDetails: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  defaultBankAccountEPayment: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  refNoTT: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  transactionDateTT: {
    type: DataTypes.STRING, // Changed from DATE to STRING
    allowNull: true,
  },
  bankNameTT: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  bankDetailsTT: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  defaultBankAccountTT: {
```

```
    type: DataTypes.STRING,
    allowNull: true,
  },
  remarksTT: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  cardType: {
    type: DataTypes.STRING, // Debit, Credit
    allowNull: true,
  },
  cardNo: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  cardHolderName: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  bankNameCard: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  bankDetailsCard: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  defaultBankAccountCard: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  discountRemarks: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  billRemarks: {
    type: DataTypes.TEXT,
    allowNull: true,
  },
  printBillOnConfirm: {
    type: DataTypes.BOOLEAN,
    defaultValue: false,
  }
}, {
```

```
    tableName: 'pr_billpaymentdetails',  
    timestamps: true,  
  });
```

```
const QRCodeModel = sequelize.define(  
  "QRCode",  
  {  
    qrid: {  
      type: DataTypes.INTEGER,  
      autoIncrement: true,  
      primaryKey: true,  
    },  
    qrimg: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    qrsecid: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    qrstatus: {  
      type: DataTypes.STRING,  
      defaultValue: "I", // Inactive  
    },  
  },  
  {  
    tableName: "qrcode",  
  }  
);
```

```
const BillServices = sequelize.define('BillServices', {  
  id: {  
    type: DataTypes.INTEGER,  
    autoIncrement: true,  
    primaryKey: true,  
  },  
  billId: {  
    type: DataTypes.INTEGER,  
    allowNull: false,  
  },  
  serviceCode: {
```

```

    type: DataTypes.STRING,
    allowNull: false,
  },
  serviceName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  serviceRate: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  concession_per: {
    type: DataTypes.DECIMAL(5, 2),
    allowNull: true,
    defaultValue: 0,
  },
  concession_amount: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: true,
    defaultValue: 0,
  },
  service_amount: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: true,
    defaultValue: 0,
  },
  totalAmount: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  netAmount: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  doctor: {
    type: DataTypes.STRING,
    allowNull: true,
  },
}, {
  alter: true,
  tableName: 'pr_billsservices',
  timestamps: true, // To include createdAt and updatedAt fields
});

```

```
const BillItems = sequelize.define('BillItems', {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  billId: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
  item_code: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  item_name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  batch_code: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  hsn_code: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  expiry_date: {
    type: DataTypes.DATE,
    allowNull: true,
  },
  rate: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  quantity: {
    type: DataTypes.INTEGER,
    allowNull: false,
    defaultValue: 1,
  },
  concession_per: {
    type: DataTypes.DECIMAL(5, 2),
    allowNull: true,
    defaultValue: 0,
  },
});
```

```

    },
    concession_amount: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: true,
      defaultValue: 0,
    },
    totalAmount: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: false,
    },
    netAmount: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: false,
    },
    doctor: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    gst: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: true,
      defaultValue: 0,
    },
  }, {
    alter: true,
    tableName: 'pr_billitems',
    timestamps: true, // To include createdAt and updatedAt fields
  });

```

```

const Advance = sequelize.define('PatientAdvance', {
  date: {
    type: DataTypes.STRING, // Date
  },
  rec_no: {
    type: DataTypes.STRING, // Receipt No.
  },
  clinic_name: {
    type: DataTypes.STRING,
    defaultValue: '1' // Clinic Name
  },
},

```

```
patient_id: {
  type: DataTypes.STRING, // Clinic Name
},
amount: {
  type: DataTypes.DECIMAL(10, 2), // Amount (stored as 'amount')
},
consume_amount: {
  type: DataTypes.DECIMAL(10, 2), // Consume Amount
},
balance_amount: {
  type: DataTypes.DECIMAL(10, 2), // Balance Amount
},
paymentMethod: {
  type: DataTypes.STRING, // Payment Mode (stored as 'paymentMethod')
},
referenceNo: {
  type: DataTypes.STRING, // Reference No. (stored as 'referenceNo')
},
chequeNo: {
  type: DataTypes.STRING, // Cheque No. (stored as 'chequeNo')
},
chequeDate: {
  type: DataTypes.STRING, // Cheque No. (stored as 'chequeNo')
},
bankName: {
  type: DataTypes.STRING, // Bank Name (stored as 'bankName')
},
referenceDate: {
  type: DataTypes.STRING, // Reference Date (stored as 'referenceDate')
},
bankDetails: {
  type: DataTypes.STRING, // Bank Details (stored as 'bankDetails')
},
defaultBankAccount: {
  type: DataTypes.STRING, // Default Bank Account (stored as 'defaultBankAccount')
},
ePaymentGatewayName: {
  type: DataTypes.STRING, // E-Payment Gateway Name (stored as
'ePaymentGatewayName')
},
ePaymentDetails: {
  type: DataTypes.STRING, // E-payment Details (stored as 'ePaymentDetails')
},
cardType: {
```



```

    type: DataTypes.STRING, // Card Type (stored as 'cardType')
  },
  cardNo: {
    type: DataTypes.STRING, // Card No. (stored as 'cardNo')
  },
  cardHolderName: {
    type: DataTypes.STRING, // Card Holder Name (stored as 'cardHolderName')
  },
  transactionDate: {
    type: DataTypes.STRING, // Card Holder Name (stored as 'cardHolderName')
  },
  approvalNo: {
    type: DataTypes.STRING, // Approval No. (stored as 'approvalNo')
  },
  printBillOnConfirm: {
    type: DataTypes.BOOLEAN, // Print Bill on Confirm (stored as 'printBillOnConfirm')
  },
  remarks: {
    type: DataTypes.TEXT, // Remarks (stored as 'remarks')
    allowNull: true
  }
}, {
  timestamps: true,
  alter: true,
  tableName: 'mainpatientadvance',
});

```

```

const CompanyAdvance = sequelize.define('CompanyAdvance', {
  date: {
    type: DataTypes.STRING, // Date
  },
  rec_no: {
    type: DataTypes.STRING, // Receipt No.
  },
  clinic_name: {
    type: DataTypes.STRING,
    defaultValue: '1' // Clinic Name
  },
  patient_id: {
    type: DataTypes.STRING, // Clinic Name
  },
  patient_name: {
    type: DataTypes.STRING, // Clinic Name
  },

```

```
company_name: {
  type: DataTypes.STRING, // Clinic Name
},
amount: {
  type: DataTypes.DECIMAL(10, 2),
  defaultValue:0.00 // Amount (stored as 'amount')
},
consume_amount: {
  type: DataTypes.DECIMAL(10, 2),
  defaultValue:0.00 // Consume Amount
},
refunded_amount: {
  type: DataTypes.DECIMAL(10, 2),
  defaultValue:0.00
},
balance_amount: {
  type: DataTypes.DECIMAL(10, 2), // Balance Amount
},
paymentMethod: {
  type: DataTypes.STRING, // Payment Mode (stored as 'paymentMethod')
},
referenceNo: {
  type: DataTypes.STRING, // Reference No. (stored as 'referenceNo')
},
chequeNo: {
  type: DataTypes.STRING, // Cheque No. (stored as 'chequeNo')
},
chequeDate: {
  type: DataTypes.STRING, // Cheque No. (stored as 'chequeNo')
},
bankName: {
  type: DataTypes.STRING, // Bank Name (stored as 'bankName')
},
referenceDate: {
  type: DataTypes.STRING, // Reference Date (stored as 'referenceDate')
},
bankDetails: {
  type: DataTypes.STRING, // Bank Details (stored as 'bankDetails')
},
defaultBankAccount: {
  type: DataTypes.STRING, // Default Bank Account (stored as 'defaultBankAccount')
},
ePaymentGatewayName: {
```

```

    type: DataTypes.STRING, // E-Payment Gateway Name (stored as
'ePaymentGatewayName')
  },
  ePaymentDetails: {
    type: DataTypes.STRING, // E-payment Details (stored as 'ePaymentDetails')
  },
  cardType: {
    type: DataTypes.STRING, // Card Type (stored as 'cardType')
  },
  cardNo: {
    type: DataTypes.STRING, // Card No. (stored as 'cardNo')
  },
  cardHolderName: {
    type: DataTypes.STRING, // Card Holder Name (stored as 'cardHolderName')
  },
  transactionDate: {
    type: DataTypes.STRING, // Card Holder Name (stored as 'cardHolderName')
  },
  approvalNo: {
    type: DataTypes.STRING, // Approval No. (stored as 'approvalNo')
  },
  printBillOnConfirm: {
    type: DataTypes.BOOLEAN, // Print Bill on Confirm (stored as 'printBillOnConfirm')
  },
  remarks: {
    type: DataTypes.TEXT, // Remarks (stored as 'remarks')
    allowNull: true
  }
}, {
  timestamps: true,
  tableName: 'maincompany_advance',
});

```

```

// sequelize.sync()
module.exports = {
  PR_patientReg,
  PR_ReferralDoc,
  PR_Appointment,
  PR_PatientVisit,
  PR_formNewCouple,
  PR_BillFindPatient,
  QRCodeModel,

```

```
BillServices,  
Advance ,  
PR_BillPaymentDetails ,  
CompanyAdvance,  
PR_BillPharmacy,  
BillItems,  
Patients  
};
```

Scripts

Start server:

```
npm start
```

Run tests:

```
npm test
```

Lint code:

```
npm run lint
```

- **CI/CD Template with GitHub Actions**

- Create `.github/workflows/node.js.yml`:

```
```yaml
```

```
name: Node.js CI
```

```
on:
```

```
 push:
```

```
 branches:
```

```
 - main
```

```
 pull_request:
```

```
 branches:
```

```
 - main
```

```
jobs:
```

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: '16'

- run: npm install

- run: npm run lint

- run: npm test

- **Create Example Test:**

In the controllers folder, create example.test.js:

```
function myAjax(formData,errorDivId,url,loadingDiv){
 // Show loading indicator
 $(loadingDiv).show();
```

```
 $.ajax({
 url: url,
 type: 'POST',
 contentType: 'application/json',
 data: JSON.stringify(formData),
 beforeSend: function() {
 // Show loading spinner or text
 $(loadingDiv).text('Loading...').show();
 },
 success: function(response) {
 // Hide loading indicator on success
 $(loadingDiv).hide();
```

```
 alert('Form submitted successfully!');
```

```

 $(errorDivId).removeClass('alert-danger').addClass('alert-success').text('Form
submitted successfully!').show();
 console.log('abc:', response);
 return response;

 },
 error: function(xhr, status, error) {
 // Hide loading indicator on error
 $(loadingDiv).hide();
 alert('Error in saving form');
 const response = JSON.parse(xhr.responseText);

 $(errorDivId).removeClass('alert-success').addClass('alert-danger').text(response.msg).
 show();
 console.error('Error:', error);
 }
});

}

```

- **APIS Structure :-**

**-> Aadhar API :-**

```

// function for genrating OTP

const generateAadhaarOtp = async (aadhaarNumber) => {
 if (!aadhaarNumber) {
 throw new Error("Aadhaar number is required");
 }

 try {
 const response = await axios.post(
 "https://kyc-api.surepass.io/api/v1/aadhaar-v2/generate-otp",
 {
 id_number: aadhaarNumber,
 },
 {

```

```

 headers: {
 "Content-Type": "application/json",
 Authorization: `Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImIhdCI6MTcyMzg
xMDUyMSwianRpljoiNzk5NGRlZTctNWZiOS00NDcyLTkxYzgtNTE4YmYxMjVjZjdhlwi
dHlwZSI6ImFjY2VzcyIsImkZSW50aXR5IjoibW50aXR5IjoibW50aXR5IjoibW50aXR5I
W8iLCJuYmYiOiE3MjM4MTA1MjEsImV4cCI6MTcyNjQwMjUyMSwiZW1haWwiOiJsaW
ZlbGlua2VyQHNIcmVwYXNzLmlvliwidGVuYW50X2lkIjoibWFpbGlzLnVzZXJfY2xhaW1zIj
p7InNjb3BlcyI6WyJlc2VyIl19fQ.LO6_zf5B0SafS0F5LAJwvjC_BoelpcxlouDupmW3J6o`
, // Authorization header with Bearer token
 },
 }
);

const { data } = response.data;

if (response.status === 200 && data.otp_sent) {
 console.log("OTP Sent successfully. Client ID:", data.client_id);
 return { clientId: data.client_id, message: "OTP sent successfully" };
} else {
 throw new Error("Failed to send OTP");
}
} catch (error) {
 console.error("Error generating OTP:", error.message);
 throw new Error("Failed to generate OTP");
}
};

```

- **function for validating OTP and Fetching Client Aadhaar Details**

```

const fetchAadhaarInfo = async (otp, clientId) => {
 if (!otp || !clientId) {
 throw new Error("OTP and Client ID are required");
 }

 try {
 const response = await axios.post(
 "https://kyc-api.surepass.io/api/v1/aadhaar-v2/submit-otp",
 {
 client_id: clientId,
 otp: otp,
 },
);
 }
};

```

```

{
 headers: {
 "Content-Type": "application/json",
 Authorization: `Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImhhdCI6MTcyMzg
xMDUyMSwianRpljoiNzk5NGRIZTctNWZiOS00NDcyLTkxYzgtNTE4YmYxMjVjZjdhliwi
dHlwZSI6ImFjY2VzcyIsImkZW50aXR5IjoiZGV2LmxpZmVsaW5rZXJAc3VyZXBhc3Mua
W8iLCJuYmYiOiE3MjM4MTA1MjEsImV4cCI6MTcyNjQwMjUyMSwiZW1haWwiOiJsaW
ZlbGlua2VyQHNIcmVwYXNzLmlvliwidGVuYW50X2lkIjoibWFpbilsInVzZXJfY2xhaW1zIj
p7InNjb3BlcyI6WyJlc2VylI19fQ.LO6_zf5B0SafS0F5LAJwvjC_Boelpcx1ouDupmW3J6o`
, // Authorization header with Bearer token
 },
}
);

```

```
const { data } = response.data;
```

```

if (response.status === 200 && data.status === "success_aadhaar") {
 console.log("Aadhaar information fetched successfully:", data);
 return data;
} else {
 throw new Error("Failed to fetch Aadhaar information");
}
} catch (error) {
 console.error("Error fetching Aadhaar info:", error.message);
 throw new Error("Failed to fetch Aadhaar information");
}
};

```

// example code to use like this

```

const aadhaarNumber = "123457897894";
const token = "x89y8sadsadjkasnd"; // Example token

```

```

// Step 1: Generate OTP
generateAadhaarOtp(aadhaarNumber)
.then((otpResponse) => {
 const clientId = otpResponse.clientId;
 const otp = "756719"; // Replace with the actual OTP received
 // Step 2: Verify OTP and fetch Aadhaar info
 return fetchAadhaarInfo(otp, clientId);
})

```



```

.then((aadhaarInfo) => {
 console.log("Aadhaar Info:", aadhaarInfo);
})
.catch((error) => {
 console.error("Error:", error.message);
});

```

- **TeliCMI API :-**

```

const axios = require("axios");

const initiateCall = async (toNumber, apiKey, token, callerId) => {
 if (!toNumber || !apiKey || !token || !callerId) {
 throw new Error("To number, API key, token, and caller ID are required");
 }

 try {
 const response = await axios.post(
 "https://rest.telecmi.com/v2/ind/click2call",
 {
 token: token, // Use the provided token
 to: toNumber, // Number to call
 extra_params: { crm: "true" }, // Example extra parameters
 callerid: callerId, // Caller ID for the call
 },
 {
 headers: {
 "Content-Type": "application/json",
 Authorization: `API-KEY ${apiKey}`, // API-KEY Authorization
 },
 }
);
 }

 const { data } = response;

 if (response.status === 200 && data.code === 200) {
 console.log(
 "Call initiated successfully:",

```

```

 data.msg,
 "Request ID:",
 data.request_id
);
 return data;
} else {
 throw new Error("Failed to initiate call");
}
} catch (error) {
 console.error("Error initiating call:", error.message);
 throw new Error("Failed to initiate call");
}
};

```

```

const toNumber = 919582711211; // Replace with the actual number to call
const apiKey = "bc233ed1-9871-4e0d-9f29-e796b8e13f58"; // Replace with actual API
key
const token =

```

```

"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJYXRIZ29yeSI6ImLuciIsImkljoiMTExMV8z
MzMzNDI2MyIsImluZXRfbm8iOiJmZmMzM0MjYzLCJuYW1lIjoiiU2FoaWwiLCJpYXQiOiE3
MjY2NTg4NzMsImV4cCI6MTcyOTI1MDg3M30uZVuiFqoWfQ9Pf9lfG2m_ygw2boE2LHh
dcN2-gXtJ2Kk"; // Replace with actual token
const callerId = 911203203882; // Replace with actual caller ID

```

```

// initiateCall(toNumber, apiKey, token, callerId)
// .then((response) => {
// console.log("Call Response:", response);
// })
// .catch((error) => {
// console.error("Error:", error.message);
// });

```

```

module.exports = { initiateCall };

```

- **Automating CI/CD with GitHub Actions**

1. Create GitHub Actions Workflow:

Navigate to your repository and create `.github/workflows/node.js.yml`.

Paste the following:

```
name: Node.js CI
```

```
on:
```

```
 push:
```

```
 branches:
```

```
 - main
```

```
 pull_request:
```

```
 branches:
```

```
 - main
```

```
jobs:
```

```
 build:
```

```
 runs-on: ubuntu-latest
```

```
 steps:
```

```
 - uses: actions/checkout@v3
```

```
 - name: Setup Node.js
```

```
 uses: actions/setup-node@v3
```

```
 with:
```

```
 node-version: '16'
```

```
 - run: npm install
```

```
 - run: npm run lint
```

```
 - run: npm test
```

2. Push to Repository:

Push your code to GitHub, and the workflow will run automatically on each push or pull request to the main branch.

- **Create README.md in the root directory and include:**

# Project Name

## Prerequisites

- Node.js >= 14.x
- npm >= 6.x

## Installation

1. Clone the repository:

```
``bash
git clone https://github.com/your-repo.git
```

2. Install dependencies:

```
npm install
```

3. Add environment variables in .env file:

```
PORT=5000
DB_URL=mysql2://localhost:3306/yourdb
```

Scripts

Start server:

```
npm start
```

Run tests:

```
npm test
```

## **Next Steps**

1. Clone and test the setup in a sample project.
2. Share the setup with your team and document the process.
3. Schedule walkthroughs or workshops for interns and new employees.