index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
```

```
    -->
    <title>Rentify</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

www

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('server:server');
var http = require('http');
```

```
/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);
```

```
  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
```

```
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

lease.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const leaseSchema = new mongoose.Schema({
```

```
tenantId: { type:Schema.Types.ObjectId, ref:"users" },
landLordId: { type:Schema.Types.ObjectId, ref:"users" },
startDate: {
  type: Date,
  required: true
},
endDate: {
  type: Date,
  required: true
},
signedDate: {
  type: Date,
  required: true
},
leaseDocument: {
  type: String,
  required: true
},
securityDepositPaidDate: {
  type: Date,
  required: false
},
propertyId: { type:Schema.Types.ObjectId, ref:"property" },
securityDepositAmount: {
  type: Number,
  required: true
},
rentPrice: {
```

```
    type: Number,
    required: true
  }
 },
 {
    versionKey: false
 });

 const leaseModel = mongoose.model('lease', leaseSchema);
 module.exports = leaseModel;
```

maintainance.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const maintenanceSchema = new mongoose.Schema({
   userId: { type:Schema.Types.ObjectId, ref:"users" },
   leaseId: { type:Schema.Types.ObjectId, ref:"lease" },
   postedDate: {
    type: Date,
    default: Date.now
   },
   updatedDate: {
    type: Date,
    default: Date.now
   },
   title: {
    type: String,
```

```
  required: true
},
description: {
  type: String,
  required: true
},
category: {
  type: String,
  required: true
},
status: {
  type: String,
  required: true
},
subCategory: {
  type: String,
  required: true
},
photos: {
  type: [String],
  required: false
},
emergency: {
  type: Boolean,
  default: false
},
entryInstructions: {
  type: String,
  required: false
```

```
    },
    entryPreferences: {
      type: String,
      required: false
    }
  },
  {
     versionKey: false
  });

  module.exports = mongoose.model('Maintenance', maintenanceSchema);
```

payments.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const paymentsSchema = new mongoose.Schema({
    userId: { type:Schema.Types.ObjectId, ref:"users" },
    amount: {
      type: Number,
      required: true
    },
    leaseId: { type:Schema.Types.ObjectId, ref:"lease" },
    cardDetails: {
      type: Object,
      required: false
    },
    bankAccount: {
      type: Object,
```

```
      required: false
    },
    paymentDate: {
      type: Date,
      required: true
    }
  },
  {
     versionKey: false
  });

  const paymentsModel = mongoose.model('payments', paymentsSchema);
  module.exports = paymentsModel;
```

property.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

var propertySchema = new Schema({
    address: { type: String, required: true },
    state: { type: String, required: true },
    city: { type: String, required: true },
    // country: {type:String,required:true},
    zip: { type: String, required: true },
    propertyType: { type: String, required: true },
    description: { type: String, required: true },
    // tourAvailability: {type:String,required:true},
    bedNo: { type: Number, required: true },
```

```
    bathNo: { type: Number, required: true },
    sqft: { type: Number, required: true },
    rentPrice: { type: Number, required: true },
    // securityDeposit: {type:Number,required:true},
    // durationfLease: {type:Number,required:true},
    contact: { type: String, required: true },
    availabilityDate: { type: Date, default: '' },
    // isAvailable: { type: Boolean, required: true },
    // petFriendly: {type:Boolean,required:true},
    // studentFriendly: {type:Boolean,required:true},
    amenities: { type: [String], default: [] },
    pictures: [{ type: String }],
    landlord: { type: Schema.Types.ObjectId, ref: "users" },
    tenant: {
      type: Schema.Types.ObjectId,
      ref: "users",
    },
    createdAt: { type: Date, default: Date.now },
    updatedAt: { type: Date, default: Date.now },
    touravailability: { type: String, required: false }
},
    {
      versionKey: false
    }
);

const propertyModel = mongoose.model('property', propertySchema);
module.exports = propertyModel;
```

user.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

var userSchema= new Schema({
    // CustomerID:{type:String,required:true},
    Email:{type:String,required:true, unique:true},
    Password:{type:String,required:true},
    FirstName:{type:String,required:true},
    LastName:{type:String,required:true},
    ImageURL:{type:String,default:""},
    DateOfBirth:{type:Date,default:""},
    City:{type:String,default:""},
    State:{type:String,default:""},
    Country:{type:String,default:""},
    Nickname:{type:String,default:""},
    PhoneNumber:{type:String,default:""},
    // addressForOrders:[{
    //     addressId: {type:String, required:true},
    //     address: {type:String, required:true},
    //     zip: {type:String, required:true}
    // }],
    // orders:[{
    //     type:Schema.Types.ObjectId,
    //     ref:"Order"
    // }],
    // About:{type: String, default: ""},
    Zip: {type: String, default:""}
},
```

```
{
   versionKey: false
});

const userModel = mongoose.model('users', userSchema);
module.exports = userModel;
```

style.css

```
body {
  padding: 50px;
  font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;
}

a {
  color: #00B7FF;
}
```

index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/h', function(req, res, next) {
  res.render('index', { title: 'Mary' });
});

module.exports = router;
```

leaseRoutes.js

```
var express = require('express');
var router = express.Router();
var Lease = require('../models/lease');
const mongoose = require('mongoose');


router.post("/create", async (req, res) => {
   try {
     // Validate the request body
     const { tenantId, landLordId, startDate, endDate, signedDate, leaseDocument, securityDepositPaidDate, propertyId, securityDepositAmount, rentPrice } = req.body;
     if (!tenantId || !landLordId || !startDate || !endDate || !signedDate || !leaseDocument || !securityDepositPaidDate || !propertyId || !securityDepositAmount || !rentPrice) {
       return res.status(400).json({ error: 'Missing required fields' });
     }

     console.log(req.body)

     // Create a new Lease object
     const newLease = new Lease({
       tenantId: mongoose.Types.ObjectId(tenantId.trim()),
       landLordId: mongoose.Types.ObjectId(landLordId.trim()),
       startDate: startDate,
       endDate:new Date(endDate),
       signedDate: new Date(signedDate),
       leaseDocument,
       securityDepositPaidDate: new Date(securityDepositPaidDate),
       propertyId: mongoose.Types.ObjectId(propertyId.trim()),
       securityDepositAmount,
```

```
      rentPrice
    });

    console.log(newLease)

  //   Save the new Lease object to the database
    const savedLease = await newLease.save();

  //   Return a success response
    res.status(201).json(savedLease);
  } catch (error) {
    // Handle any errors that occur
    console.error(error);
    res.status(500).json({ error: 'Internal server error' });
  }
});


router.get("/get/:leaseId", async (req, res) => {
  try {
    const lease = await Lease.findById(req.params.leaseId);
    if (!lease) {
      return res.status(404).json({ error: 'Lease not found' });
    }
    res.json(lease);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});
```

```
router.put("/update/:leaseId", async (req, res) => {
    try {
        const lease = await Lease.findByIdAndUpdate(req.params.leaseId, req.body, {
new: true });
        if (!lease) {
            return res.status(404).json({ error: 'Lease not found' });
        }
        res.json(lease);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

router.delete("/delete/:leaseId", async (req, res) => {
    try {
        const lease = await Lease.findByIdAndDelete(req.params.leaseId);
        if (!lease) {
            return res.status(404).json({ error: 'Lease not found' });
        }
        res.json(lease);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

router.get("/get/all", async (req, res) => {
    try {
```

```
        const leases = await Lease.find();
        console.log(leases)
        res.json(leases);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

router.get("/get/user/tenant/:tenantId", async (req, res) => {
    try {
        const leases = await Lease.find({ tenantId: req.params.tenantId });
        res.json(leases);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

router.get("/get/user/landlord/:landlordId", async (req, res) => {
    try {
        const leases = await Lease.find({ landLordId: req.params.landlordId });
        res.json(leases);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

router.get("/get/property/:propertyId", async (req, res) => {
    try {
        const leases = await Lease.find({ propertyId: req.params.propertyId });
```

```
        res.json(leases);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

module.exports = router;
```

paymentRoutes.js

```
var express = require('express');
var app = express.Router();
var Payment = require('../models/payments');

app.post('/create', async (req, res) => {
    try {
      const newPayment = new Payment(req.body);
      await newPayment.save();
      res.status(201).json(newPayment);
    } catch (err) {
      res.status(400).json({ message: err.message });
    }
  });

app.get('/get/all', async (req, res) => {
  try {
    const payments = await Payment.find();
    res.json(payments);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
```

```
});


app.get('/get/:id',async (req, res) => {
   try {
     const payment = await Payment.findById(req.params.id);
     if (!payment) {
       return res.status(404).send();
     }
     res.send(payment);
   } catch (error) {
     res.status(500).send(error);
   }
 });


 app.put('/update/:id', async (req, res) => {
   try {
     const payment = await Payment.findByIdAndUpdate(req.params.id, req.body, {
       new: true,
       runValidators: true,
     });
     if (!payment) {
       return res.status(404).send();
     }
     res.send(payment);
   } catch (error) {
     res.status(400).send(error);
   }
```

```
});


app.delete('/delete/:id', async (req, res) => {
  try {
    const payment = await Payment.findByIdAndDelete(req.params.id);
    if (!payment) {
      return res.status(404).send();
    }
    res.send(payment);
  } catch (error) {
    res.status(500).send(error);
  }
});


module.exports = app;
```

propertyRoutes.js

```
var express = require('express');
var router = express.Router();
var Property = require('../models/property');


router.post("/create", (req, res) => {
    const property = new Property(req.body);
    console.log(req.body)
    property.save((err, newProperty) => {
        if (err) {
```

```
        console.log(err)
          res.status(400).send(err);
        } else {
          console.log(newProperty)
          res.status(201).send(newProperty);
        }
    });
  });


router.get("/get/all", (req, res) => {
    Property.find((err, properties) => {
      if (err) {
        res.status(400).send(err);
      } else {
        console.log('hii')
        res.status(200).send(properties);
      }
    });
  });

  // Get a single property by ID
  router.get("/get/:id",(req, res) => {
    Property.findById(req.params.id, (err, property) => {
      if (err) {
        res.status(400).send(err);
      } else if (!property) {
        res.status(404).send({ message: 'Property not found' });
      } else {
```

```
      res.status(200).send(property);
    }
  });
});

router.get("/find/:query", (req, res) =>{
  const { query } = req.params;
  console.log('inside the find', query)
// Perform the search query on the property collection
Property.find({
  $or: [
    { city: { $regex: query, $options: 'i' } },
    { zip: { $regex: query, $options: 'i' } },
  ],
})
  .then((properties) => {
    console.log('found these', properties)
    res.json(properties);
  })
  .catch((error) => {
    console.error(error);
    res.status(500).json({ error: 'An error occurred' });
  });
})

// Update a property by ID
router.put("/update/:id", (req, res) => {
  Property.findByIdAndUpdate(req.params.id, req.body, { new: true }, (err,
updatedProperty) => {
```

```
    if (err) {
      res.status(400).send(err);
    } else if (!updatedProperty) {
      res.status(404).send({ message: 'Property not found' });
    } else {
      res.status(200).send(updatedProperty);
    }
  });
});

// Delete a property by ID
router.delete("/delete/:id", (req, res) => {
  Property.findByIdAndDelete(req.params.id, (err, deletedProperty) => {
    if (err) {
      res.status(400).send(err);
    } else if (!deletedProperty) {
      res.status(404).send({ message: 'Property not found' });
    } else {
      res.status(204).send();
    }
  });
});

router.get("/get/tenant/:tenantId", async (req, res) => {
  try {
      const properties = await Property.find({ tenant: req.params.tenantId });
      // res.json(properties);
      res.status(200).send(properties)
  } catch (err) {
```

```
        res.status(400).json({ error: err.message });
    }
});

router.get("/get/landlord/:landlordId", async (req, res) => {
  console.log(req.params.landlordId)
    try {
        const leases = await Property.find({ landlord: req.params.landlordId });
        console.log('leases',leases)
        // res.json(leases);
        res.status(200).send(leases)

    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

module.exports = router;
```

users.js

```
var express = require('express');
var router = express.Router();
var User = require('../models/User');
var bcrypt = require('bcryptjs');
var salt = bcrypt.genSaltSync(10);

/* GET users listing. */
router.get('/', function (req, res, next) {
```

```
  res.send('respond with a resource');
});

router.post("/create/profile", (req, res) => {
console.log("reqqqqq",req.body)
  const newUser = User({
    Email: req.body.emailId,
    FirstName: req.body.firstName,
    LastName: req.body.lastName,
    Password: bcrypt.hashSync(req.body.password, salt),
    PhoneNumber: req.body.phoneNumber,
  })

  newUser.save().then((document) => {
    console.log("User Saved successfully." + document);
    res.status(200).send(document)
  },
    (err) => {
      console.log("Unable to save user",err)
      res.status(401).send("Error Creating user")
    }
    )

})

router.post("/login/check", (req, res) => {
  console.log(req.body);
  const { emailId, password } = req.body;
```

```
  User.findOne({
    Email: req.body.emailId
  }, (err, user) => {
    if (err) {
      console.log('err', err);
    }
    else if (user) {
      console.log('cust', user);
      if (!bcrypt.compare(req.body.password, user.Password)) {
        console.log('Invalid Credentials');
        res.status(401).end('Invalid crentials');
      }
      else {
        //  const payload = { _id: customer._id, emailId: customer.Email };
        //  const token = jwt.sign(payload, secret)
        //  console.log('result'+customer);
        const loginResponse = {
          //  token: "JWT " + token,
          user
        }
        res.status(200).send(loginResponse);
      }
    }
  })

})

module.exports = router;
```

dbConfig.js

```
require('dotenv').config()


const config = {
    // mongoDB: process.env.DB_Connection_String
    mongoDB    :
"mongodb+srv://HouseRentalManagementApplication:deejaynavnikSJSU@cluster0.s
comigl.mongodb.net/?retryWrites=true&w=majority"
};


module.exports = config;
```

error.jade

```
extends layout


block content
  h1= message
  h2= error.status
  pre #{error.stack}
```

index.jade

```
extends layout


block content
  h1= title
  p Welcome to #{title}
```

layput.jade

```
doctype html
```

```
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
const mongoose = require('mongoose');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var maintenanceRouter = require('./routes/maintenanceRoutes');
var leaseRouter = require('./routes/leaseRoutes');
var paymentsRouter = require('./routes/paymentRoutes');
var propertyRouter = require('./routes/propertyRoutes');

var app = express();


//Allow Access Control
app.use(function(req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
  res.setHeader('Access-Control-Allow-Credentials', 'true');
```

```
  res.setHeader('Access-Control-Allow-Methods',
'GET,HEAD,OPTIONS,POST,PUT,DELETE');
  res.setHeader('Access-Control-Allow-Headers', 'Access-Control-Allow-Headers,
Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method,
Access-Control-Request-Headers');
  res.setHeader('Cache-Control', 'no-cache');
  next();
});

//MongoDB connection Part
const { mongoDB } = require('./utils/dbConfig');

var options = {
    useNewUrlParser: true,
    useUnifiedTopology: true,
}

mongoose.connect(mongoDB, options, (err, res) => {
  console.log(mongoDB);
    if(err){
        console.log(err);
        console.log('MongoDB connection failed');
    }
    else {
        console.log('MongoDB connected');
    }
})



// view engine setup
```

```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/maintenance', maintenanceRouter);
app.use('/lease', leaseRouter);
app.use('/payments', paymentsRouter);
app.use('/property', propertyRouter);

// // 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) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
```

```
  res.render('error');
});

app.listen(3001, () => {
  console.log("Server Listening on port 3001")
})

module.exports = app;
```