Backend App Setup:

```
1   //require modules
2   const express = require('express');
3   const morgan = require('morgan');
4   const methodOverride = require('method-override');
5   const mongoose = require('mongoose');
6   const session = require('express-session');
7   const MongoStore = require('connect-mongo');
8   const flash = require('connect-flash');
9   const propertyRoutes = require('./routes/propertyRoutes');
10  const userRoutes = require('./routes/userRoutes');
11  const requestRoutes = require('./routes/requestRoutes');
12
13  //create app
14  const app = express();
15
16  //configure app
17  let port = 3000;
18  let host = 'localhost';
19  app.set('view engine', 'ejs');
20  const mongUri = 'mongodb+srv://admin:admin123@cluster0.ick0s.m
21
22  //connect to MongoDB
23  mongoose.connect(mongUri)
24  .then(() => {
25      //start the server
26      app.listen(port, host, ()=>{
27          console.log('Server is running on port', port);
28      });
29  })
30  .catch(err => console.log(err.message));
31
32  //mount middleware
33  app.use(
34      session({
35          secret: "ajfeirf90aeu9eroejfoefj",
36          resave: false,
37          saveUninitialized: false,
38          store: new MongoStore({mongoUrl: mongUri}),
39          cookie: {maxAge: 60*60*1000}
40          })
41  );
42  app.use(flash());
43
44  app.use((req, res, next) => {
45      //console.log(req.session);
46      res.locals.user = req.session.user || null;
47      res.locals.errorMessages = req.flash('error');
48      res.locals.successMessages = req.flash('success');
49      next();
50  });
51
52  app.use(express.static('public'));
53  app.use(express.urlencoded({extended: true}));
54  app.use(morgan('tiny'));
55  app.use(methodOverride('_method'));
```

These lines of code initialize the web application using Express Node and a MongoDBAtlas database. The first 14 lines of code set up constants to require packages and routes. Lines 16-20 configure the app to be viewed. Lines 23-41 connect the application to the database and create a user session. Lines 42-50 setup for flash messages to be used. Lines 52-55 tell the app to use the packages named in the first block of code.

```
57  //set up routes
58  app.get('/', (req, res) => {
59      res.render('index');
60  });
61
62  app.use('/properties', propertyRoutes);
63  app.use('/users', userRoutes);
64  app.use('/requests', requestRoutes);
65
66  app.use((req, res, next) => {
67      let err = new Error('The server cannot locate ' + req.url);
68      err.status = 404;
69      next(err);
70  });
71
72  app.use((err, req, res, next) => {
73      console.log(err.stack);
74      if (!err.status) {
75          err.status = 500;
76          err.message = ("Internal Server Error")
77      }
78
79      res.status(err.status);
80      res.render('error', {error: err});
81  });
```

Lines 58-64 direct the app to use specific routes. Lines 66-81 prep the app for error handling.

```
1   const express = require('express');
2   const controller = require('../controllers/propertyController');
3   const {upload} = require('../models/property');
4   const {isLoggedIn, isSeller} = require('../middlewares/auth');
5   const {validateId} = require('../middlewares/validator')
6
7   const router = express.Router();
8
9
10  router.get('/', controller.index);
11
12  router.get('/new', controller.new);
13
14  router.post('/', controller.create);
15
16  router.get('/:id', controller.show);
17
18  router.get('/:id/apply', controller.apply);
19
20  router.post('/:id/apply', controller.submitApplication);
21
22  router.get('/:id/edit', controller.edit);
23
24  router.put('/:id', controller.update);
25
26  router.delete('/:id', controller.delete);
27
28  module.exports = router;
```

These lines of code direct routes from ‘/properties’ to their relevant controllers.

```
const express = require('express');
const controller = require('../controllers/requestController');
//const {isLoggedIn, isSeller} = require('../middlewares/auth');
//const {validateId} = require('../middlewares/validator')

const router = express.Router();

router.get('/', controller.index);

router.post('/', controller.create);

module.exports = router;
```

These lines of code direct routes from ‘/requests’ to their relevant controllers. Although the requests function is not fully functional thus far.

```
const express = require('express');
const controller = require('../controllers/userController');
const {isGuest, isLoggedIn} = require('../middlewares/auth');

const router = express.Router();

router.get('/new', isGuest, controller.new);

router.get('/contact', controller.contact);

router.post('/', isGuest, controller.create);

router.get('/login', isGuest, controller.getUserLogin);

router.post('/login', isGuest, controller.login);

router.get('/dashboard', controller.profile);

router.get('/logout', isLoggedIn, controller.logout);

router.get('/application/:id', controller.update);

router.post('/application/:id', controller.submitUpdate);

module.exports = router;
```

These lines of code direct routes from ‘/users’ to their relevant controllers.

```
const model = require('../models/property');
const tenant = require('../models/tenant');
const user = require('../models/user');

exports.index = (req, res, next)=>{
    model.find()
    .then(properties => res.render('./property/index', {properties}))
    .catch(err => next(err));
};
```

These lines of code set up constants for the property controller and direct users to the property index page.

```
exports.new = (req, res)=>{
    res.render('./property/new');
};


exports.create = (req, res, next) => {
    console.log('Form Data Received:', req.body);

    let property = new model(req.body);

    if(property.bedroom == 'One'){
        property.bath = 'One';
        property.squareFootage = '800';
        property.images = '/images/oneBedroom.jpg';
    } else if(property.bedroom == 'Two'){
        property.bath = 'Two';
        property.squareFootage = '1100';
        property.images = '/images/twoBedroom.jpg';
    } else if(property.bedroom == 'Three'){
        property.bath = 'Two';
        property.squareFootage = '1400';
        property.images = '/images/threeBedroom.jpg';
    }

    // Save the property
    property.save()
    .then((property) => {
        console.log('Saved Property:', property);
        req.flash('success', 'You have successfully added property');
        res.redirect('/properties');
    })
    .catch(err => {
        console.error('Save Error:', err);
        if(err.name === 'ValidationError') {
            err.status = 400;
            console.error('Validation Errors:', err.errors);
        }

        if(err.code === 11000) {
            req.flash('error', 'Unit number has been used');
            return res.redirect('/properties/new');
        }

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

These lines of code direct users to the page that them to create new properties in the database, and once the user has submitted the form the create function will add any remaining required information and save it into the database.

```
exports.show = (req, res, next)=>{
    let id = req.params.id;
    if(!id.match(/^[0-9a-fA-F]{24}$/)) {
        let err = new Error('Invalid item id');
        err.status = 400;
        return next(err);
    }

    model.findById(id)
    .then((property) => {
        if(property) {
            user.findById(req.session.user)
            .then(currentUser => {
                res.render('./property/show', {property, user: currentUser, userType: user.userType });
            })

        } else {
            let err = new Error('Cannot find a item with id ' + id);
            err.status = 404;
            next(err);
        }
    })
    .catch(err => next(err));


};
```

These lines of code query the database to locate the specifically requested property then render the property details page.

```
exports.apply = (req, res, next)=>{
    let id = req.params.id;
    if(!id.match(/^[0-9a-fA-F]{24}$/)) {
        let err = new Error('Invalid item id');
        err.status = 400;
        return next(err);
    }


    model.findById(id)
    .then(property => {
        if(property) {

            user.findById(req.session.user)
            .then(currentUser => {
                if(currentUser) {
                    res.render('./property/apply', {property, user: currentUser});
                } else {
                    let err = new Error('Cannot find user information');
                    err.status = 404;
                    next(err);
                }
            })
            .catch(err => next(err));
        } else {
            let err = new Error('Cannot find a property with id ' + id);
            err.status = 404;
            next(err);
        }
    })
    .catch(err => next(err));
};

exports.submitApplication = (req, res, next) => {

    let application = new tenant({
        unitNumber: req.body.propertyId,
        tenant: req.session.user,
        moveInDate: req.body.moveInDate,
        employmentStatus: req.body.employmentStatus,
        annualIncome: req.body.annualIncome,
        additionalNotes: req.body.additionalNotes
    });

    application.save()
    .then(application => {
        req.flash('success', 'Application submitted successfully');
        res.redirect('/properties');
    })
    .catch(err => {
        if(err.name === 'ValidationError') {
            err.status = 400;
        }
        next(err);
    });
};
```

These lines of code allow users to apply for a specific property, and once application form is submitted it links the user to a new tenant document in the database.

```
142  exports.edit = (req, res, next)=>{
143      let id = req.params.id;
144
145      model.findById(id)
146      .then(property => {
147          if(property) {
148              res.render('./property/edit', {property});
149          } else {
150              let err = new Error('Cannot find a item with id ' + id);
151              err.status = 404;
152              next(err);
153          }
154      })
155      .catch(err => next(err));
156
157  };
158
159  exports.update = (req, res, next) => {
160      let id = req.params.id;
161
162      model.findByIdAndUpdate(id, req.body,
163          {
164              runValidators: true,
165              new: true
166          })
167          .then(property => {
168              if (property) {
169                  req.flash('success', 'You have successfully updated property');
170                  res.redirect('/users/dashboard');
171              } else {
172                  let err = new Error('Cannot find a property with id ' + id);
173                  err.status = 404;
174                  next(err);
175              }
176          })
177          .catch(err => {
178              if (err.name === 'ValidationError') {
179                  err.status = 400;
180                  console.error('Validation Errors:', err.errors);
181              }
182
183              if (err.code === 11000) {
184                  req.flash('error', 'Unit number has been used');
185                  return res.redirect('/properties/new');
186              }
187
188              next(err);
189          });
190  };
```

These line of code allow for updating the properties in the database.

```
1   const model = require('../models/user');
2   const property = require('../models/property');
3   const tenant = require('../models/tenant');
4   const request = require('../models/request');
5
6   exports.new = (req, res)=>{
7       res.render('./user/new');
8   };
9
10  exports.contact = (req, res)=>{
11      res.render('./user/contact');
12  };
13
14  exports.create = (req, res, next)=>{
15      let user = new model(req.body);
16      user.save()
17      .then(user=> {
18          req.flash('success', 'You have successfully registered');
19          res.redirect('/users/login')
20      })
21      .catch(err=>{
22          if(err.name === 'ValidationError' ) {
23              req.flash('error', err.message);
24              return res.redirect('/users/new');
25          }
26
27          if(err.code === 11000) {
28              req.flash('error', 'Email has been used');
29              return res.redirect('/users/new');
30          }
31
32          next(err);
33      });
34  };
```

Lines 1-4 require the user controller to use the schemas as named constants within the controller. Lines 6-12 will render the new user creation page and the property contact page. Lines 14 - 34 will save the new user into the database.

```
36  exports.getUserLogin = (req, res, next) => {
37      res.render('./user/login');
38  }
39
40  exports.login = (req, res, next)=>{
41
42      let email = req.body.email;
43      let password = req.body.password;
44      model.findOne({ email: email })
45      .then(user => {
46          if (!user) {
47              console.log('wrong email address');
48              req.flash('error', 'wrong email address');
49              res.redirect('/users/login');
50              } else {
51              user.comparePassword(password)
52              .then(result=>{
53                  if(result) {
54                      req.session.user = user._id;
55                      req.flash('success', 'You have successfully logged in');
56                      res.redirect('/');
57              } else {
58                  req.flash('error', 'wrong password');
59                  res.redirect('/users/login');
60              }
61              });
62          }
63      })
64      .catch(err => next(err));
65  };
```

Lines 36-38 will render the login page for the user. Then lines 40-65 will process that login attempt and alert that the login credentials are incorrect and redirect back to the login page for another attempt.

```
exports.profile = (req, res, next)=>{
    let id = req.session.user;


    model.findById(id)
        .then(user => {
            if (!user) {
                req.flash('error', 'User not found');
                return res.redirect('/users/login');
            }


            if (user.userType === 'management' || user.userType === 'maintenance') {
                Promise.all([
                    tenant.find({}).populate('tenant'),
                    property.find({}),
                    request.find({}),
                    model.find({})
                ])
                .then(([tenants, properties, requests, users]) => {
                    res.render('./user/managementDashboard', {
                        currentUser: user,
                        tenants,
                        properties,
                        requests,
                        users,
                        userType: user.userType
                    });
                })
                .catch(err => next(err));
            }

            else if (user.userType === 'potential' || user.userType === 'tenant') {
                tenant.findOne({ tenant: id })
                    .then(tenantInfo => {
                        if (tenantInfo) {
                            property.findById(tenantInfo.unitNumber)
                                .then(propertyInfo => {
                                    res.render('./user/dashboard', {
                                        user,
                                        tenantInfo,
                                        propertyInfo,
                                        userType: user.userType
                                    });
                                })
                                .catch(err => next(err));
                        } else {
                            res.render('./user/dashboard', {
                                user,
                                tenantInfo: null,
                                propertyInfo: null
                            });
                        }
                    })
                    .catch(err => next(err));
            }

            else {
                req.flash('error', 'Invalid user type');
                return res.redirect('/');
```

These lines of code will read the user type and then render the required dashboard and information for that user.

```
165  exports.submitUpdate = (req, res, next) => {
166
167      let application = new tenant({
168          unitNumber: req.body.propertyId,
169          tenant: req.session.user,
170          moveInDate: req.body.moveInDate,
171          employmentStatus: req.body.employmentStatus,
172          annualIncome: req.body.annualIncome,
173          additionalNotes: req.body.additionalNotes
174      });
175
176      application.save()
177      .then(application => {
178          req.flash('success', 'Application submitted successfully');
179          res.redirect('/properties');
180      })
181      .catch(err => {
182          if(err.name === 'ValidationError') {
183              err.status = 400;
184          }
185          next(err);
186      });
187  };
188
189  exports.logout = (req, res, next)=>{
190      req.session.destroy(err=>{
191          if(err)
192              return next(err);
193          else {
194              res.redirect('/');
195          }
196      });
197
198  };
```

Lines 165-187 will allow for user information to be updated. Lines 189-198 allow for user to effectively logout.